

## 2.8) SQL-Datenbankzugriff mit PHP

Um auf eine MySQL-Datenbank zuzugreifen, muss zuerst eine Verbindung (**`mysqli_connect`**) hergestellt werden. Dazu ist der **Host**, auf dem der MySQL-Server läuft, der **Benutzername** sowie das **Kennwort** anzugeben. Anschließend muss noch eine Datenbank auf dem DBS ausgewählt werden (**`mysqli_select_db`**). Danach können die SQL-Anweisungen folgen (**`mysqli_query`**). Im Falle einer SQL-Abfrage liefert diese Funktion alle Tupel zurück, welche der Reihe nach mit **`mysqli_fetch_array`** abgearbeitet werden können.

Im Folgenden soll als Beispiel ein Gästebuch realisiert werden. Dazu wurde eine Tabelle **meldung** erstellt, welche die einzelnen Nachrichten aufnehmen soll:

```
CREATE TABLE meldung (  
  id INT NOT NULL AUTO_INCREMENT,  
  datum DATETIME DEFAULT NULL,  
  name VARCHAR(200) DEFAULT NULL,  
  eintrag TEXT,  
  CONSTRAINT PK_id PRIMARY KEY (id)  
);
```

Zusätzlich zu den Daten **datum**, **name** und **eintrag** wurde ein Attribut **id** eingeführt, welches als Schlüssel dient.

Das folgende Skript **show.php** liest alle Einträge der Datenbank aus und gibt sie in Tabellenform aus:

```
<?php  
$DBHost = "127.0.0.1";  
$DBUser = "root";  
$DBPasswd = "";  
$DBName = "db";  
  
//Verbindung zu DB-Server herstellen  
$con=mysqli_connect($DBHost, $DBUser, $DBPasswd, $DBName)  
    OR die("Konnte DB-Server nicht erreichen!");  
mysqli_select_db($con,$DBName);  
?  
<html>  
  <head>  
    <title>Alle Meldungen</title>  
  </head>  
  <body>  
    <?php  
      $erg=mysqli_query($con, "SELECT datum, name, eintrag FROM meldung  
ORDER BY datum DESC");  
      echo mysqli_error($con);  
  
      while($row=mysqli_fetch_array($erg))  
      {  
        echo "<table bgcolor=\"#dddddd\" border=\"0\" width=\"600\">
```

```

\n";
        echo "<tr><td>Datum: </td><td>".$row["datum"]."</td></tr> \n";
        echo "<tr><td>Name:</td><td>".$row["name"]."</td></tr> \n";
        echo "<tr><td valign=\"top\">Eintrag:</td> \n";
        echo "<td>".nl2br(htmlentities($row["eintrag"]))."</td></tr>
\n";
        echo "</table>\n<br>\n";
    }
?>
<hr><a href="insert.php">neuen Eintrag hinzufügen</a>
</body>
</html>

```

Datum: 2018-11-15 20:19:20

Name: Prof. Andreas Lahmer

Eintrag: Hier sehen sie die SQL-Anbindung mittels PHP in Form eines Gästebuchs!

---

[neuen Eintrag hinzufügen](#)

Um neue Einträge für das Gästebuch zu erstellen, existiert ein weiteres Skript insert.php:

```

<?php
$DBHost = "127.0.0.1";
$DBUser = "root";
$DBPasswd = "";
$DBName = "db";

//Verbindung zu DB-Server herstellen
$con=mysqli_connect($DBHost, $DBUser, $DBPasswd)
    OR die("Konnte DB-Server nicht erreichen!");
mysqli_select_db($con,$DBName);
?>
<html>
<head>
    <title>Neuer Eintrag in unser Gästebuch</title>
</head>
<body>
    <?php

        if(isset($_GET["submit"]))
        {
            $submit = $_GET["submit"];
            $name = $_GET["name"];
            $eintrag = $_GET["eintrag"];

            //Der Submit-Button wurde gedrückt --> die Werte müssen
            überprüft werden
            //und bei Gültigkeit in die DB eingefügt werden

```

```

        $DatenOK=1;           //wir gehen prinzipiell von der
Gültigkeit der Daten aus
        $error="";           //es gab noch keine Fehlermeldung bis hier
hier

        if($name=="")        //Kein Name eingegeben
        {
            $DatenOK=0;
            $error.="Es muss ein Name eingegeben werden!<br>\n";
        }
        if($eintrag=="")     //Kein Kommentar eingegeben
        {
            $DatenOK=0;
            $error.="Ein Eintrag ohne Kommentar macht nicht viel
Sinn!<br>\n";
        }
        if($DatenOK)         //Daten OK -> also in DB eintragen
        {
            $timestamp=date("Y-m-d h:i:s",
time());
            mysqli_query($con,"INSERT INTO eintraege (datum, name,
eintrag) VALUES (\"$timestamp\", \"$name\", \"$eintrag\" );");
            echo mysqli_error($con);
            echo "<b>Daten wurden eingetragen.<b>";
        }
        else
        {
            echo "<h2>Fehler: </h2>\n"; //Fehlermeldung
            echo $error;
        }
    }

    //Formular
?>

    <form action="insert.php" method="GET">
        Name: <input type="text" name="name" size="30" maxlength="200"
value=""><br>
        Text: <br><textarea rows="10" cols="50" wrap="virtual"
name="eintrag"></textarea>
        <br><input type="submit" name="submit" value="Absenden">
    </form>
    <a href="show.php"> Alle Einträge anzeigen</a>
</body>
</html>

```

Name:

Text:

Absenden

[Alle Einträge anzeigen](#)

From:

<http://elearn.bgamstetten.ac.at/wiki/> - **Wiki**

Permanent link:

[http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf8bi\\_201819:2:2\\_08](http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf8bi_201819:2:2_08)

Last update: **2019/01/17 10:11**



# DATEIBEHANDLUNG

Programme speichern ihre Informationen in Variablen. Leider bleiben diese immer nur bis zum nächsten Stromausfall oder Betriebssystemabsturz erhalten. Und wenn das Programm verlassen wird, ist ihr Inhalt ebenfalls Geschichte. Damit Sie auf Ihre Daten auch morgen noch kraftvoll zugreifen können, empfiehlt es sich, diese in einer Datei abzulegen. Dazu können Sie die Daten als Ausgabestrom in die Datei schreiben. Das Vorgehen entspricht dem bei der Bildschirmausgabe per `cout`. Diese Form wird sequenziell genannt, weil die Daten nacheinander in der Reihenfolge, wie sie geschrieben wurden, in der Datei landen. Sie können aber auch einen Datenblock an eine beliebige Stelle der Datei schreiben. Später können Sie diesen Datenblock wieder zurückholen, indem Sie den internen Dateizeiger an diese Stelle positionieren und den Datenblock wieder lesen. Diese Vorgehensweise ist typisch für Klassen, insbesondere, wenn sie in irgendeiner Form sortiert abgelegt werden sollen.

## fstream

Über die Headerdatei `fstream` wird die Funktionalität zum Lesen und Schreiben von Dateien zur Verfügung gestellt. Soll nur geschrieben werden, dann kann auch die Headerdatei `ofstream` genutzt werden. Ebenso kann, wenn nur gelesen werden soll, als Header `ifstream` zum Einsatz kommen. Der übliche Ablauf zum Lesen oder Schreiben von Dateien ist folgender:

- Objekt zum Lesen oder Schreiben im Programm anlegen
- Objekt mit dem Dateinamen zum öffnen verknüpfen
- Text über das Objekt schreiben oder lesen
- Datei schliessen

## Schreiben einer Datei

In Zeile 5 wird ein Objekt angelegt, dass ähnlich wie `cout` oder `cin` die Ausgabe in die Datei übernimmt. Der Name des Objekts kann beliebig vergeben werden, so lange dieser noch nicht verwendet wird. Die Datei `beispiel.txt` wird in Zeile 6 geöffnet und im aktuellen Verzeichnis angelegt, falls diese noch nicht existiert. Nun da die Datei geöffnet wurde, kann in Zeile 7 etwas in die Datei geschrieben werden. Zum Schluss muss noch die Datei geschlossen werden, damit alle gepufferten Schreibvorgänge abgeschlossen werden (Zeile 8).

```
1  #include <fstream>
2  using namespace std;
3
4  int main () {
5      ofstream fileout;
6      fileout.open("beispiel.txt");
7      fileout << "Das steht jetzt in der Datei.";
8      fileout.close();
9      return 0;
10 }
```

## Lesen einer Datei

Nun wollen wir den gerade geschriebenen Inhalt der Datei wieder auslesen und auf dem Bildschirm anzeigen.

```
1  #include <iostream>
2  #include <fstream>
3  #include <string>
4  using namespace std ;
5
6  int main() {
7      string dateizeile;
8      ifstream fin;
9      fin.open("beispiel.txt");
10     getline(fin,dateizeile);
11     fin.close () ;
12     cout << "Zeile in der Datei: " << dateizeile << endl ;
13     return 0;
14 }
```

Wir benötigen einen String `dateizeile`, in dem wir die gelesene Zeile aus der Datei aufbewahren können. Eine komplette Zeile lässt sich mittels der Funktion `getline` einlesen. Es wird das mit der Datei verknüpfte Objekt und der String, in den diese Zeile gespeichert werden, übergeben (Zeile 10). Nach dem Schliessen der Datei wird der gelesene String in Zeile 12 ausgegeben.

From:

<http://elearn.bgamstetten.ac.at/wiki/> - Wiki

Permanent link:

[http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf7bi\\_201718:3\\_cplusplus:3\\_07](http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf7bi_201718:3_cplusplus:3_07)

Last update: **2018/01/31 14:44**



# Doppelt verkettete Listen

Doppelt verkettete Listen (oder doubly linked lists) sind häufig benutzte Datenstrukturen und eine Verallgemeinerung der einfach verketteten Listen, die ich hier anhand von Beispielen in der Programmiersprache C++ vorstellen will.

Sie funktioniert in jeder Programmiersprache genauso, die Zeiger oder Referenzen und so etwas wie Klassen oder Structs zur Verfügung stellt.

## Wozu?

Doppelt verkettete Listen, oder „double linked lists“ braucht man immer dann, wenn man sich in einer Liste leicht vorwärts und rückwärts bewegen können muss, und wenn man schnell und einfach Elemente der Liste an beliebigen Positionen löschen und neue einfügen muss.

Denn einfach verkettete Listen haben den Nachteil, dass man sie nur in einer Richtung durchlaufen kann.

Darüber hinaus kann man ein referenziertes Listenelement nicht unmittelbar löschen, da man von diesem Element keinen Zugriff auf das Vorgängerelement hat.

Doppelt verkettete Listen umgehen dieses Problem, indem sie in jedem Knoten zusätzlich noch eine Referenz auf den Vorgängerknoten speichern.

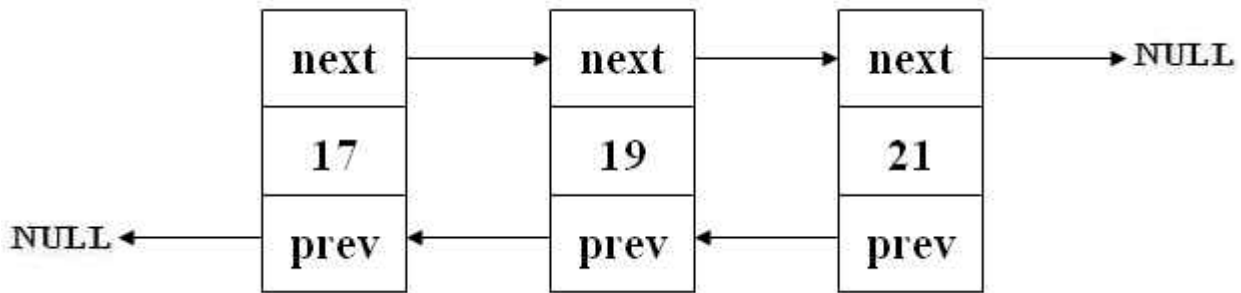
## Die Grundidee

Im Gegensatz zu einfach verketteten Listen haben doppelt verkettete Listen in den Knoten eine zusätzliche Instanzvariable für die Referenz auf den Vorgängerknoten

Die Idee einer doppelt verketteten Liste ist es also, für jedes Element zwei Zeiger zu speichern, einen nach links und einen nach rechts:

```
struct DList{
    DList*left;
    DList *right;
    int data;
};
```

Die eigentlichen Daten, die in der Liste gespeichert werden sollen, stehen in data, das hier als int definiert ist. Jeder andere Datentyp, z.B. ein Zeiger auf beliebige andere Strukturen, funktioniert genau so.



## Einfügen, Löschen und Ausgeben von Elementen

```

#include <iostream>

using namespace std;

struct DListe{
    DListe *next;
    DListe *prev;
    int zahl;
};

int main(int argc, char** argv) {

    DListe *head=NULL;           //Anlegen des Head-Pointers
    DListe *tail=NULL;           //Anlegen des Tail-Pointers
    DListe *help=NULL;           //Anlegen des Help-Pointers (für das Löschen
und Ausgeben)

    //Einfügen von 10 Elementen in eine doppelt verkettete Liste
    for(int i=0;i<10;i++)
    {
        DListe *elem=new DListe();           //Element erzeugen
        elem->zahl=i;                         //Attribut zahl befüllen
        if(head==NULL && tail==NULL)         //noch kein Element in der Liste
        {
            head=elem;                       //Head zeigt auf das neue Element
            tail=elem;                       //Tail zeigt auf das neue Element
            elem->next=NULL;                  //Next zeigt auf NULL
            elem->prev=NULL;                  //Prev zeigt auf NULL
        }
        else                                 //Elemente sind bereits vorhanden &
        Element wird am Ende eingefügt
        {
            tail->next=elem;                  //tail->next auf das neue Element
            elem->prev=tail;                  //elem->prev zeigt auf das
ursprünglich letzte Element
            elem->next=NULL;                  //elem->next zeigt auf NULL
        }
    }
}
  
```



```
        tail=elem;                                //tail zeigt auf das neue letzte
Element
    }
}

    cout << "Geben Sie an, welches Element (0-9) Sie loeschen wollen!" <<
endl;
    int loesche=0;
    cin >> loesche; //zB. Element mit der zahl=2 wird gelöscht

    help=head;                                    //help zeigt auf head (Anfang der Liste)
    while(help->zahl!=loesche)                    //Wenn help->zahl!=loesche, dann wandert
help in der Liste weiter
    {
        help=help->next;
    }
    //Überprüfe nochmals mit Ausgabe, ob beim richtigen Element (=2)
angekommen?
    cout << help->zahl;
    //Sonderfall 1. Element soll gelöscht werden (zahl=0)
    if(help==head)
    {
        head->next->prev=NULL;
        head=head->next;
        head->prev=NULL;
    }
    else if(help==tail) //Sonderfall letztes Element soll gelöscht werden
    {
        tail=help->prev;
        help->prev->next=help->next;
    }
    else //Element ist zwischen 2 anderen Elementen
    {
        help->prev->next=help->next;
        help->next->prev=help->prev;
    }

    //Ausgabe der doppelt verketteten Liste
    cout << endl << endl;
    help=head;
    while(help!=NULL)
    {
        cout << help->zahl << " -> ";
        help=help->next;
    }

    return 0;
}
```

## Sortiertes Einfügen

```

do{
    DListe *elem=new DListe();
    cout << endl << "Geben Sie ein Element ein, das sie einfuegen
wollen!" << endl;
    cin >> elem->zahl;
    elem->next=NULL;
    elem->prev=NULL;
    help=head;

    while((elem->zahl>help->zahl) && (help->next!=NULL))
    {
        help=help->next;
    }

    if(head==help)    //1. Fall => Element wird an erster Stelle
eingefügt
    {
        elem->next=help;
        help->prev=elem;
        head=elem;
    }
    else if(tail==help) //2. Fall => Element wird an letzter Stelle
eingefügt
    {
        elem->prev=help;
        help->next=elem;
        tail=elem;
    }
    else    //3. Fall => Element wird in der Mitte, also zwischen 2
anderen Elemente eingefügt
    {
        elem->next=help;
        help->prev->next=elem;
        elem->prev=help->prev;
        help->prev=elem;
    }

    //Ausgabe der Doppelt verketteten Liste
    ausgabe(head);

    cout << "Wollen Sie noch ein Element einfuegen (j/n)" << endl;
}while(getch()=='j');
```

From:

<http://elearn.bgamstetten.ac.at/wiki/> - **Wiki**

Permanent link:

[http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf8bi\\_201819:3:3\\_05](http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf8bi_201819:3:3_05)



Last update: **2019/03/25 16:05**