technische universität
dortmund

Klausurdeckblatt

Matrikel – Nr.:

Bitte tragen Sie Ihre Matrikelnummer und Ihren Namen in die dafür vorgesehenen Felder ein. Bitte in deutlicher Handschrift mit einem schwarzen Stift (nicht Bleistift).
Das Feld mit dem **Barcode ist unbedingt frei zu lassen.**
Danke.

Vorname:

Nachname:

Der Bereich unterhalb dieser Linie kann von der Fakultät frei gestaltet werden.

Einführung in die Programmierung

07. Februar 2024

Prof. Dr.-Ing. Peter Ulbrich

Studiengang: ☐ ET/IT ☐ IKT ☐ Physik ☐ WiMa ☐ _____

Hinweise:
Sie haben zur Bearbeitung dieser Klausur drei volle Zeitstunden zur Verfügung (d.h. 1 Punkt ≡ 3 min). Notieren Sie Ihre Lösungen direkt auf den ausgeteilten Aufgabenblättern unter Verwendung eines dokumentenechten, schwarzen oder blauen Stifts. Als Hilfsmittel ist ausschließlich ein Wörterbuch (ohne handschriftliche Ergänzungen o.ä.) zulässig. Zusätzliche Papierbögen können Sie von der Aufsicht beziehen. Die Aufsicht darf Ihnen keinerlei Hilfestellung inhaltlicher Art geben.

Für die Aufgaben der Klausur können Sie die folgenden Punkte erhalten:

Aufgabe	Erreichbare Punkte	Erzielte Punkte
1	10	
2	10	
3	10	
4	10	
5	10	
6	10	
Summe	60	

Note: _____

Bevor Sie mit der Bearbeitung der Aufgaben beginnen, tragen Sie auf **allen** Blättern Ihre Matrikelnummer ein. Sie sollten sich zuerst alle Aufgaben durchlesen und dabei auf die von Ihnen geforderten Ergebnisse achten, bevor Sie versuchen, die Aufgaben zu lösen. **Viel Erfolg!**

Diese Seite wurde absichtlich leer gelassen.

Aufgabe 1: ***Programmverständnis*** [10 Punkte]
a) *[4 Punkte]*
Analysieren Sie nachfolgendes Programm. Welche Zahlen werden in welcher Reihenfolge ausgegeben?

```
#include <iostream>
using namespace std;

void f(int &i) {
  i = i - 2;
}

int main(){
  int i = 6, k = 14;

  do {
    switch (i) {
    case 6:
    case 1:
      cout << i + k << endl;
      break;
    case 2:
      cout << i << endl;
    case 5:
      cout << i + 1 << endl;
      break;
    default:
      cout << k + 3 * i << endl;
    }

    f(k);
    --i;

  } while (k > 0);

  return 0;
}
```

b) *[6 Punkte]* Analysieren Sie nachfolgendes Programm. Wie lautet die Ausgabe?

```
#include <iostream>
#include <cctype>
using namespace std;

class MachtNix {
private:
  char c;
public:
  MachtNix(char c) {
    this->c = c;
    cout << "+" << c << endl;
  }
  ~MachtNix() {
    cout << "-" << c << endl;
  }
};

MachtNix x('G');

int main() {
  MachtNix x('M');
  for (int k = 0; k < 2; k++) {
    MachtNix x('F' + k);
    if (k > 0) {
      MachtNix x('i');
    }
  }
  MachtNix y('Y');

  return 0;
}
```

Aufgabe 2: ***Typsystem und Speicherverwaltung*** [10 Punkte]

a) Datentypen *[3 Punkte]*
Folgende Beispiele zeigen die Herausforderungen bei der Nutzung von Datentypen. Beschreiben Sie zu jeder Funktion kurz das Problem der vorliegenden Implementierung. Erweitern Sie im Anschluss die `main()` im Sinne einer defensiven Programmierung so, dass die Probleme nicht mehr auftreten.

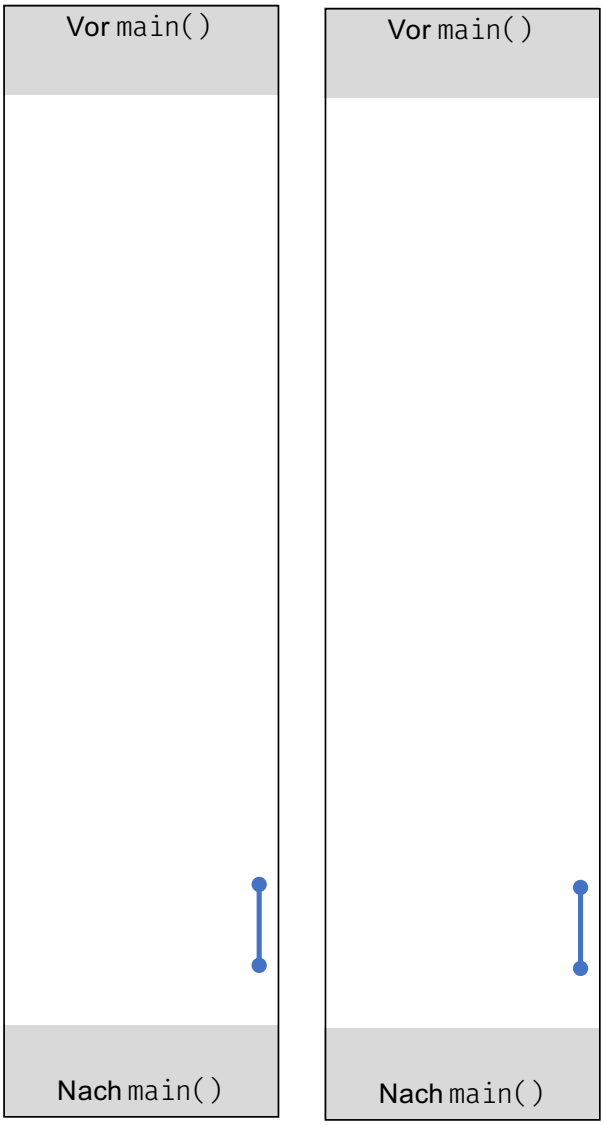
```
#include <iostream>

// Beispiel 1: Nehme an i = 1
int add(int i) {
    short result = 32767 + i;
    return result;
}
// Beispiel 2: Nehme an a = -10
bool greater5(int a) {
    unsigned int b = 5;
    if (a > b) {
        return true;
    } else {
        return false;
    }
}
// Beispiel 3: Nehme an i = 0
unsigned int dec(unsigned int i) {
    i--;
    return i;
}
```

Kurze Beschreibung:

```
int main() {
    // Lese Daten ein
    int x, y, z;
    std::cin >> x >> y >> z;









    // Problem 1
    std::cout << add(x) << "\n";
    // Problem 2
    std::cout << greater5(y) << "\n";
    // Problem 3
    std::cout << dec(z) << "\n";

    return 0;
}
```

b) Sichtbarkeit und Lebensspanne *[2 Punkte]*
Geben Sie die Lebensspanne (Wann?) und Sichtbarkeit (Wo?) für jede Variable/Objekt an. Markieren Sie die Spanne(n) in der jeweiligen Spalte durch einen vertikalen Strich (siehe Beispiel für `i`).

```
class MyClass {
    int x;
  public:
    static int y;
    MyClass(int value) : x(value) {
        y++;
    }
};

int MyClass::y = 0;
MyClass g(19);

char someFunc(short a, short b) {
    static unsigned short c = 0;
    return a + b + c;
}

int main() {
    MyClass l(42);
    int j = 15;

    for (int i = 0; i < 10; i++) {
        someFunc(i, j);
    }
}
```

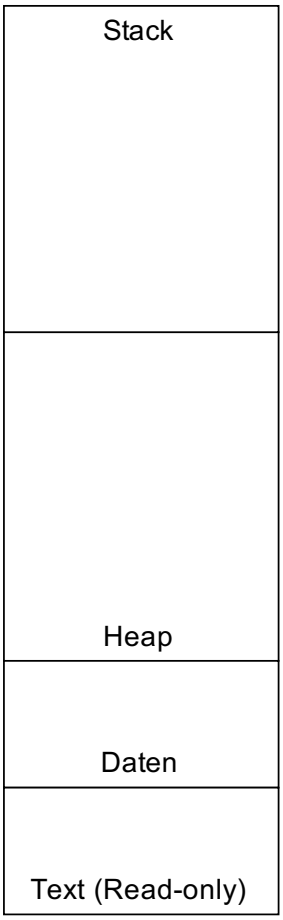


Platz für Notizen:

c) Speichersegmente und Variablen *[3 Punkte]*
In der Vorlesung haben Sie die Grundlagen der Speicherverwaltung in C++ kennengelernt. Im folgenden ist ein Programm und ein Speicherlayout (rechts) vorgegeben. Ordnen Sie **alle Variablen** dem Segment zu, in welchem sie durch den Compiler bzw. die Laufzeitumgebung abgelegt werden. Wo werden die **Übergabeparameter** und der **Rückgabewert** von `myAdd()` abgelegt? Geben Sie weiterhin an, wo der eigentliche **Programmcode** gespeichert wird.

Hinweis: Schreiben Sie die Namen in die korrekten Segmente oder zeichnen Sie Pfeile aus dem Code.

```
#include "MyClass.h"

int x = 10;
const double pi = 3.14;
char z = 'A';

int myAdd(int op1, int op2) {
    int c = x + 5;
    return op1 + op2 + c;
}


int main() {
    int a = 5;
    const float b = 2.71;

    MyClass m_1(42);
    MyClass* m_2 = new MyClass(99);

    int d = myAdd(3, 4);

    delete m_2;

    return d;
}
```



Platz für Notizen:

d) Speicherverwaltung *[2 Punkte]*
Folgendes Beispiel nutzt dynamische Speicherallokation (`new`). Beschreiben Sie kurz das Problem dieser Implementierung und korrigieren Sie den Code.

```
#include <iostream>
#include <vector>

class MyClass {
  public:
    MyClass(int value) : data(new int(value)) {}
    ~MyClass() {
        delete data;
    }

    int* data;
};

MyClass* myFunction() {
    MyClass* myObject = new MyClass(42);
    return myObject;
}

int main() {
    char userInput;
    std::vector<MyClass*> myClassObjects;

    std::cout << "Press 'y' to continue: ";

    do {
        std::cin >> userInput;
        myClassObjects.push_back(myFunction());
    } while (userInput != 'y');

    for (const auto& obj : myClassObjects) {
      std::cout << obj->data;


    }
    return 0;
}
```

Kurze Beschreibung:

Aufgabe 3: *Zeiger* [10 Punkte]
a) *[4 Punkte]* Gegeben sei folgendes Programm, welches (eventuell etwas umständlich und übertrieben) von Zeigern Gebrauch macht:

```
#include <iostream>
using namespace std;

int main() {
    int a   = 3;
    int *pa = &a;
    int b   = *pa - 1;
    cout << "A) " << a << ", " << b << ", " << *pa << endl;

    int c   = 11;
    int d   = 8;
    int *pc = &c;
    int *pd = &d;
    *pc     = 7;
    d      -= 2;
    c      += 5;
    *pc     = *pd;
    cout << "B) " << c << ", " << d << ", " << *pc << ", " << *pd;
    cout << endl;

    short e   = 2;
    short f[] = {5, 7, 9, 11};
    *(f+1)    = *f - e;
    e         = f[2];
    (*f)++;
    cout << "C) " << f[0] << ", " << f[1] << ", " << f[2] << ", "
                  << f[3] << ", " << e    << endl;

    long g[] = {90, 70, 50, 30};
    long *p  = &g[2] - 1;
    *(p+2)   = *p - 2;
    cout << "D) " << g[0] << ", " << g[1] << ", " << g[2] << ", "
                  << g[3] << ", " << *p   << endl;

    return 0;
}
```

Welche Ausgabe erzeugt das gegebene Programm?

A)

B)

C)

D)

Platz für Notizen:

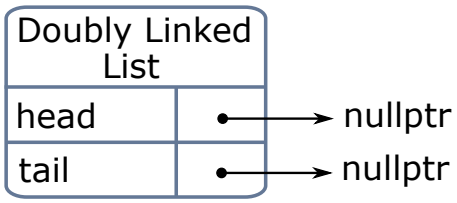
b) *[6 Punkte]*
Die Klasse `DoublyLinkedList` repräsentiert eine doppelt verkettete Liste, die Werte vom Typ `int` speichert. Jedes Listenelement enthält einen Zeiger auf den Vorgänger und den Nachfolger. Sie ist wie folgt definiert:

```
#include <iostream>
using namespace std;

class DoublyLinkedList{
private:
    struct Element{
        int data;
        Element *next;        // Zeiger auf das naechste Element
        Element *prev;        // Zeiger auf das vorherige Element
    };
    Element *head;            // Zeiger auf erstes Element
    Element *tail;            // Zeiger auf letztes Element

public:
    DoublyLinkedList()  { head = tail = nullptr; }
    ~DoublyLinkedList() { clear(); }
    bool empty()        { return (head == nullptr && tail == nullptr); }
    void clear();
    void removeFirst();       // zu implementieren
};
```

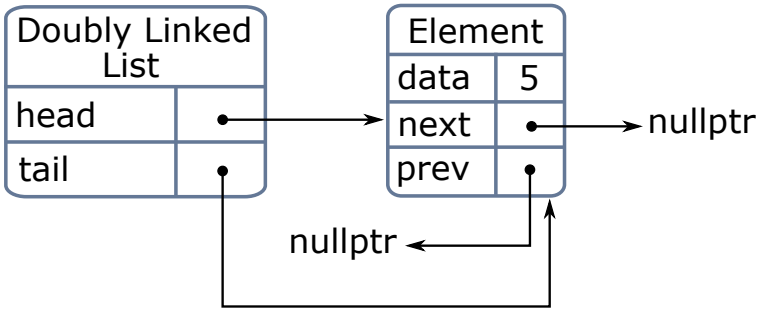
Ihre Aufgabe:
Implementieren Sie die Methode `void DoublyLinkedList::removeFirst()`, die das erste Element am Anfang der Liste entfernt. Beachten Sie dabei, dass das Element am Anfang der Liste nur entfernt werden kann, wenn die Liste nicht leer ist. Geben Sie in diesem Fall ebenfalls den Wert des entfernten Elements mit `cout` aus. Verwenden Sie zur Bearbeitung dieser Aufgabe den vorgegebenen Coderahmen nach den Beispielen.

Beispiel:
Die nachfolgenden Abbildungen veranschaulichen die Struktur und Funktionsweise einer doppelt verketteten Liste.

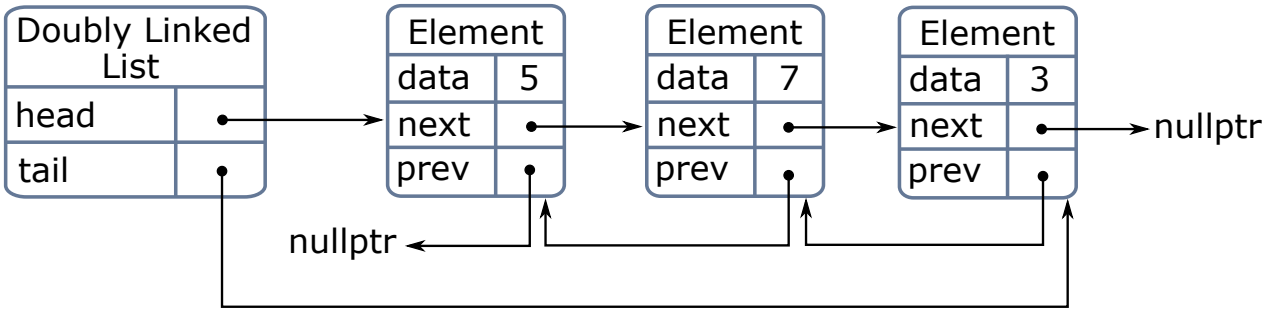
Doppelt verkettete leere Liste:



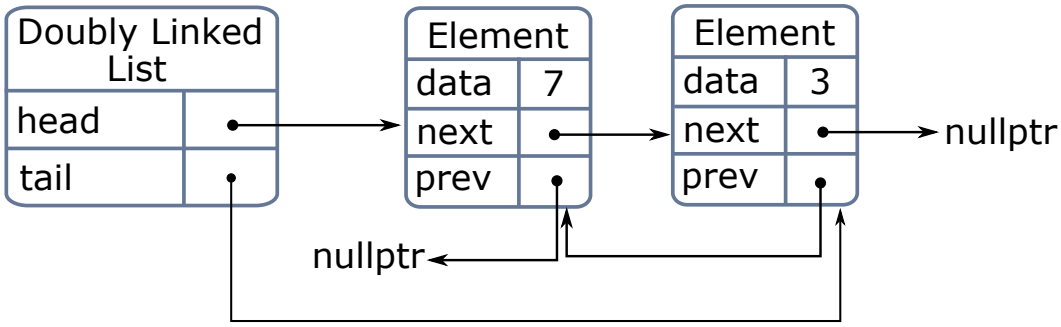
Doppelt verkettete Liste mit einem Element:



Doppelt verkettete Liste mit drei Elementen:



Nach dem Aufruf von `removeFirst()` sieht die Beispiel-Liste wie folgt aus:



```
void DoublyLinkedList::removeFirst() {
  // Fuegen Sie hier Ihren Code ein









} // Ende von DoublyLinkedList::removeFirst()
```

Aufgabe 4: ***Rekursion*** [10 Punkte]

a) *[4 Punkte]* Eine Bank bietet das Sparprogramm *Kapital987* an. Dabei zahlt man ein Mal 987 Euro auf einem Sparkonto ein. Danach werden dem Konto jährlich Zinsen zu einem Zinssatz von $p \in \mathbb{R}$ gutgeschrieben. Jedoch werden jährlich Gebühren in Höhe von zwei Prozent des Guthabens fällig, wenn das Guthaben 1500 Euro übersteigt. Für das Guthaben nach n Jahren ergibt sich damit

$$K_n = \begin{cases} 987 & \text{, wenn } n = 0, \\ (1+p) \cdot K_{n-1} - \frac{2 \cdot K_{n-1}}{100} & \text{, wenn } n > 0 \text{ und } K_{n-1} > 1500 \\ (1+p) \cdot K_{n-1} & \text{, sonst.} \end{cases}$$

Implementieren Sie eine Funktion `double guthaben(unsigned int n, double p)`, die für gegebenen Zinssatz $p > 0.0$ rekursiv das Guthaben nach n Jahren nach obiger Vorschrift berechnet und zurückgibt.

```
double guthaben(unsigned int n, double p) {
  // Fuegen Sie hier Ihren Code ein









} // Ende von double guthaben(unsigned int n, double p)
```

b) *[6 Punkte]*
Gegeben sei die Implementierung eines binären Suchbaums für `int`-Werte, die hier nur auszugsweise dargestellt ist. (Funktionsweise wie in der Vorlesung.)

Implementieren Sie die private Methode `levelProd`, die auf rekursive Weise das Produkt der Werte (`data`) aller Knoten berechnet, die *genau* die Tiefe `depth` haben. Die Tiefe eines Knotens gibt die Anzahl der Kanten bis zur Wurzel an. Die Wurzel hat Tiefe 0, die Kinder der Wurzel haben Tiefe 1, und so weiter. Ein leerer Baum hat keine Knoten.

```
#include <iostream>
using namespace std;

class BinTree {
private:
    struct Node {
        int data;
        Node *left, *right;
    } *root;
    Node *insert(Node *node, int value);
    int levelProd(Node *node, int depth);   // zu implementieren
public:
    BinTree()                  { root = nullptr; }
    void insert(int value)     { root = insert(root, value); }
    int levelProd(int depth)   { return levelProd(root, depth); }
    ~BinTree();
};
```

```
int BinTree::levelProd(Node *node, int depth) {
// Fuegen Sie hier Ihren Code ein









 // Ende von BinTree::levelProd(Node *node, int depth)
```

Zusatzblatt für Lösungen zur Aufgabe 4

Aufgabe 5: ***Klassen*** [10 Punkte]
a) **Polymorphie** *[4 Punkte]*

```
#include <iostream>
using namespace std;

class Bla {
public:
    Bla(int i) {
        cout << " Bla(" << i << ")";
    }
    virtual ~Bla() {
        cout << " ~Bla()";
    }
    virtual void foo() {
        cout << " Bla::foo()";
    }
};

class Blub : public Bla {
public:
    Blub(int i) : Bla(-i) {
        cout << " Blub(" << i << ")";
    }
    virtual ~Blub() {
        cout << " ~Blub()";
    }
    virtual void foo() {
        cout << " Blub::foo()";
    }
};

int main()
{
    cout << "x: ";
    Bla *x = new Blub(2);
    x->foo();
    cout << endl << "y: ";
    Blub y (3);
    y.foo();
    {
        cout << endl << "z: ";
        Bla z(7);
        z.foo();
        delete x;
    }
    cout << endl << "ende: ";
    return 0;
}
```

Was gibt die `main`-Funktion des Programms aus?

`x:`

`y:`

`z:`

`ende:`

Platz für Notizen:

b) **Klassenhierarchie** *[6 Punkte]*
Auf den folgenden Seiten befinden sich Gerüste für die Klassen `Koerper`, `Quader` und `Zylinder`, die die Berechnung der Dichte von Körpern ermöglichen sollen. Ergänzen Sie diese an den markierten Stellen, wie in den Teilaufgaben b_1 - b_6 gefordert.
Hinweis: Die Implementierung der Methoden soll *inline*, also direkt in der Deklaration der Klasse erfolgen.

b_1) Erweitern Sie die Klasse `Koerper` um ein privates Attribut `m`, das die Masse des Körpers repräsentieren soll (vom Typ `double`). Versehen Sie die Klasse mit einem geeigneten Konstruktor, der das Attribut initialisiert.

b_2) Legen Sie eine **rein virtuelle** Methode mit dem Namen `getVolumen` in der Klasse `Koerper` an, die keine Parameter enthält und einen `double` Wert zurückgibt.

b_3) Implementieren Sie eine Methode `getDichte` in der Klasse `Koerper`, die die Dichte des Körpers berechnet. Dies geschieht indem Sie die Masse (gegeben durch das Attribut `m`) durch das Volumen (gegeben durch die virtuelle Methode `getVolumen`) teilen. Der Rückgabewert der Methode soll den Typ `double` haben.

b_4) Implementieren Sie die Klasse `Quader`, die die Klasse `Koerper` spezialisiert. Ein Quader soll drei private Attribute `l`, `b` und `h` vom Typ `double` besitzen, die für Länge, Breite und Höhe des Quaders stehen. Initialisieren Sie diese Attribute durch die Implementierung eines passenden Konstruktors.

b_5) Erweitern Sie die Klasse `Quader` und implementieren Sie die virtuelle Methode `getVolumen` so, dass sie das Volumen des Quaders liefert. Dieses berechnet sich aus $l * b * h$.

b_6) Implementieren Sie die dritte Klasse `Zylinder` so, dass diese zwei private Attribute `r` (Radius) und `h` (Höhe) vom Typ `double` enthält. Implementieren Sie auch hier wieder einen passenden Konstruktor sowie die Methode `getVolumen`. Das Volumen eines Zylinders berechnet sich durch $\pi * r^2 * h$. Gehen Sie davon aus, dass Ihnen die Konstante `M_PI` für π aus der Mathe-Bibliothek zur Verfügung steht.

```
#include <cmath>

class Koerper {
  private:
    // Attribute anlegen (b1):




  public:
    // Konstruktor anlegen & Attribute initialisieren (b1):









    // Virtuelle Methode 'getVolumen' anlegen (b2):









    // Methode 'getDichte' anlegen (b3):









};
```

```
// Hier die Klasse Quader implementieren (b4+b5):
```

```
// Hier die Klasse Zylinder implementieren (b6):
```

Aufgabe 6: ***Schablonen & Ausnahmen*** [10 Punkte]
a) *[5.5 Punkte]*
Die unten angegebene Klasse stellt eine Schlüssel-Wert-Tabelle bereit. Mit ihrer Hilfe können Werte zu einem Schlüssel hinterlegt und später wieder abgerufen werden.

```
class SchluesselWertTabelle {
    public:
    int *schluessel;
    string *werte;
    bool *belegt;
    int kapazitaet = 0;

    SchluesselWertTabelle(int n) : kapazitaet(n) {
        schluessel = new int[n];
        werte = new string[n];
        belegt = new bool[n];
    }
    ~SchluesselWertTabelle() {
        delete[] schluessel;
        delete[] werte;
        delete[] belegt;
    }
    string entfernen(int schl, int &fehlercode) {
        for (int i = 0; i < kapazitaet; i++) {
            if (belegt[i] && schluessel[i] == schl) {
                belegt[i] = false;
                return werte[i];
            }
        }
        fehlercode = 1;
        return string();
    }
    void einfuegen(int schl, string wrt, int &fehlercode) {
        int free_index = -1;
        for (int i = 0; i < kapazitaet; i++) {
            if (!belegt[i]) free_index = i;
            if (belegt[i] && schluessel[i] == schl) {
                fehlercode = 2;
                return;
            }
        }
        if (free_index != -1) {
            schluessel[free_index] = schl;
            werte[free_index] = wrt;
            belegt[free_index] = true;
        } else fehlercode = 3;
    }
};
```

Die Klasse ist bisher nur für Schlüssel vom Typ `int` und Werte vom Typ `std::string` implementiert.

a_1) Ändern Sie die Klasse zu einer Klassen-Schablone mit zwei Typ-Parametern (Vorzugsweise als S und W oder K und V benannt), die Schlüssel-Werte Paare von beliebigen Datentypen verwaltet. Verwenden Sie dafür den vorgegebenen Code auf den folgenden Seiten. Passen Sie diesen direkt an, indem Sie **Code hinzufügen** oder bestehenden **Code durchstreichen und ersetzen**.

```
// den folgenden Code anpassen:


class SchluesselWertTabelle {

    public:

    int *schluessel;

    string *werte;

    bool *belegt;

    int kapazitaet = 0;

    SchluesselWertTabelle(int n) : kapazitaet(n) {

        schluessel = new int[n];

        werte = new string[n];

        belegt = new bool[n];

    }

    ~SchluesselWertTabelle() {

        delete[] schluessel;

        delete[] werte;

        delete[] belegt;

    }

    string entfernen(int schl, int &fehlercode) {

        for (int i = 0; i < kapazitaet; i++) {

            if (belegt[i] && schluessel[i] == schl) {
```

```
                belegt[i] = false;
                return werte[i];
            }
        }
        fehlercode = 1;
        return string();
    }
    void einfuegen(int schl, string wrt, int &fehlercode) {
        int free_index = -1;
        for (int i = 0; i < kapazitaet; i++) {
            if (!belegt[i]) free_index = i;
            if (belegt[i] && schluessel[i] == schl) {
                fehlercode = 2;
                return;
            }
        }
        if (free_index != -1) {
            schluessel[free_index] = schl;
            werte[free_index] = wrt;
            belegt[free_index] = true;
        } else fehlercode = 3;
    }
};

// Ende Aufgabe (a1)
```

a_2) Ergänzen Sie die folgende `main`-Funktion, sodass zuerst ein Objekt der Klasse `SchluesselWertTabelle` mit Namen `tab` und Kapazität von drei erzeugt wird.
Typ der Schlüssel soll `std::string` sein und Werte sollen vom Typ `int` sein. Fügen Sie in `tab` drei Paaren aus nicht-leeren `std::string`s und `int`s ein. Entfernen Sie anschließend eines dieser Paare wieder und geben den Wert auf der Konsole aus. Prüfen Sie nach jedem einfügen und entfernen auf mögliche Fehlercodes und behandeln diese gegebenenfalls durch eine Ausgabe auf der Konsole und Beendigung des Programms.

```
#include <iostream>
using namespace std;
#include "schluesselwerttabelle.h"  // Ihre Antwort aus 6a1

// Fehlercodes der Tabelle als Hilfsfunktion
void print_fehlercode(int fehlercode) {
    switch (fehlercode) {
        case 1:
        cerr << "Nicht gefunden" << endl; break;
        case 2:
        cerr << "Schon vorhanden" << endl; break;
        case 3:
        cerr << "Tabelle voll" << endl; break;
    }
}









int main(){
```

```
    return 0;
}

// Ende Aufgabe (a2)
```

b) *[4.5 Punkte]*
Die Klasse aus der vorherigen Teilaufgabe arbeitet mit einfachen Fehlercodes. In dieser Teilaufgabe passen Sie die Methoden der Klasse `SchluesselWertTabelle` an, indem Sie die Fehlerbehandlung mit Fehlercodes durch eine Ausnahmebehandlung mit *C++-Exceptions* ersetzen. Nutzen Sie dazu eine sinnvolle Anzahl an Klassen zur differenzierten Fehlerbehandlung und erhalten Sie die ursprüngliche Funktionalität. Passen Sie dabei die Signaturen und Rückgabewerte der Methoden an. Die Tabelle selbst sollen sie diesmal *nicht* in eine Schablone umwandeln.
Zusätzlich zur modifizierten Tabelle sollen Sie in einer `main`-Methode eine Tabelle anlegen, in diese ein Paar einfügen und anschließend wieder entfernen. Dabei müssen sie eine Ausnahmebehandlung implementieren.
Verwenden Sie dafür den vorgegebenen Code auf den folgenden Seiten. Passen Sie diesen direkt an, indem Sie **Code hinzufügen** oder bestehenden **Code durchstreichen und ersetzen**.

```
// A6 b)
#include <iostream>
using namespace std;

// Hier die Klassenhierarchie der Exceptions einfuegen.
```

```
// Fehlercodes durch Exceptions ersetzen.
class SchluesselWertTabelle {
    // Attribute und Konstruktoren sind wie in Teil a) gegeben

    void einfuegen(int schl, string wrt, int &fehlercode) {
        int free_index = -1;

        for (int i = 0; i < kapazitaet; i++) {

            if (!belegt[i]) free_index = i;
            if (belegt[i] && schluessel[i] == schl) {

                fehlercode = 2; // Schon vorhanden

                return;

            }

        }
        if (free_index != -1) {
            schluessel[free_index] = schl;

            werte[free_index] = wrt;

            belegt[free_index] = true;

        } else {

            fehlercode = 3; // Tabelle Voll

        }
    }
    string entfernen(int schl, int &fehlercode) {
        for (int i = 0; i < kapazitaet; i++) {

            if (belegt[i] && schluessel[i] == schl) {
                belegt[i] = false;

                return werte[i];

            }

        }

        fehlercode = 1; // Nicht gefunden

        return string();
    }
};
```

```
int main() {

    return 0;
}
// Ende Aufgabe 6b
```