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

20. März 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_1) *[2 Punkte]* Wandeln Sie den unten angegeben `if`-Ausdruck in der Funktion `zeichen` in einen äquivalenten `switch/case`-Ausdruck um.

```
char zeichen(int x) {
    char c;
    if (x == 5) c = 'a';
    else if (x == 1 || x == 2) c = 'b';
    else if (x == 3) c = 'y';
    else c = 'z';
    return c;
}
```

```
char zeichen(int x) {
    // Ihre Loesung hier einfuegen









}
```

a_2) *[2 Punkte]* In der Vorlesung haben Sie die elementaren Datenstrukturen Keller/Stapel (`Stack`) und Schlange (`Queue`) kennengelernt. Geben Sie an, welche Ausgabe von dem folgenden Programm erzeugt wird.

```
#include <iostream>
#include <Stack.h>
#include <Queue.h>
using namespace std;

int main() {
    Stack<int> s;
    Queue<int> q;
    for (int i = 0; i < 10; i++) s.push(i);
    for (int j = 0; j < 10; j++) q.enq(j);
    for (int k = 0; k < 3; k++) {
        q.deq();
        s.push(q.front());
    }
    s.pop();
    cout << s.top() << "␣" << q.front() << endl;
    return 0;
}
```

Ausgabe: _____

b) *[6 Punkte]* Geben Sie für das unten angegebene Programm an, welche Ausgabe erzeugt wird.

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

class MachtNix;
static MachtNix* KannNix = nullptr;

class MachtNix {
    private:
    char mType;
    int mId;
    public:
    MachtNix(char aType) : mType(aType) {
        static int sCnt = 0;
        mId = sCnt++;
        cout << '+' << mType << mId << endl;
    }
    void TutNix(char c) {
        if (KannNix == nullptr) {
            KannNix = new MachtNix(c);
        } else {
            delete KannNix;
            KannNix = nullptr;
        }
    }
    ~MachtNix() {
        cout << '-' << mType << mId << endl;
    }
};

MachtNix x('G');

int main() {
    MachtNix x('H');
    for (int i = 0; i < 2; i++) {
        MachtNix x('S');
        x.TutNix('D');
    }
    cout << "Ist␣Nix?␣" << (int)(KannNix==0) << endl;
    return 0;
    cout << "Tut␣Nix" << endl;
}
```

Ausgabe:

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.

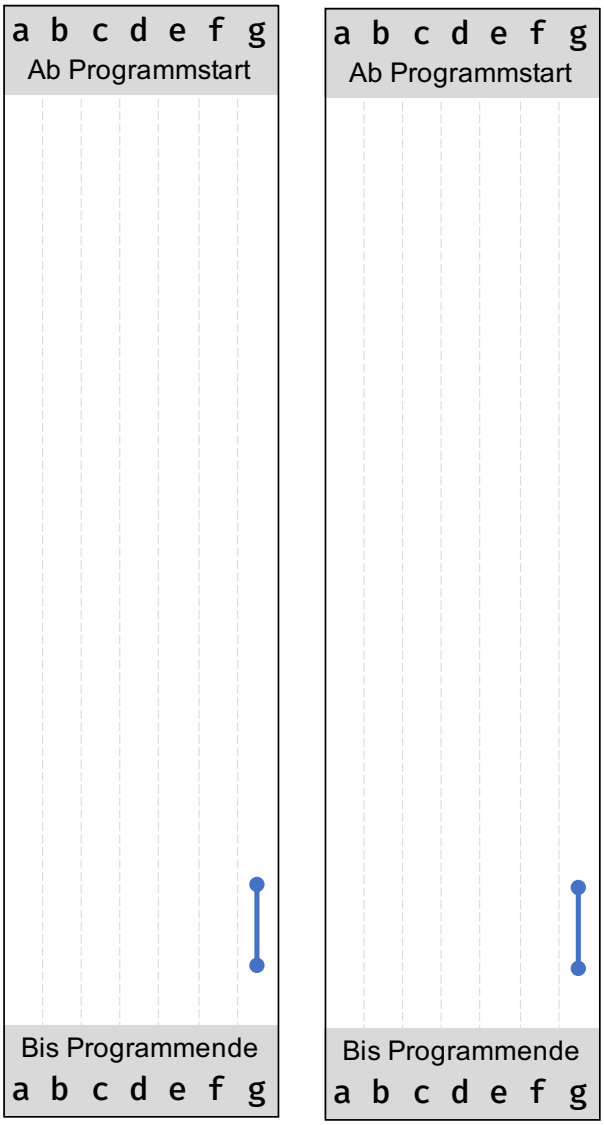
```
#include <iostream>

// Beispiel 1
bool isGreater(int a,
               unsigned int b) {
    if (a > b) {
        return true;
    } else {
        return false;
    }
}
// Beispiel 2
unsigned short lapCounter() {
    static unsigned short laps = 0;
    return laps++;
}
// Beispiel 3
unsigned short dec(int i) {
    short result = 4711 - i;
    return result;
}
```

Kurze Beschreibung:

```
int main() {
    // Lese Daten fuer die Funktionen ein
    int x, y, z;
    std::cin >> x >> y >> z;
    // HIER EIGENE LOESUNG ERGAENZEN









    // Problem 1
    std::cout << isGreater(x) << "\n";
    // Problem 2
    for (int j=0; j <= y; j++) { std::cout << lapCounter() << "\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 a;
    MyClass(int value) : x(value) {
        a++;
    }
};

int MyClass::a = 0;
MyClass b(19);

char someFunc(short c) {
    static unsigned short d = 0;
    return c + d++;
}

int main() {
    MyClass e(42);
    int f = 15;

    for (int g = 0; g < 10; g++) {
        someFunc(f);
    }
}
```



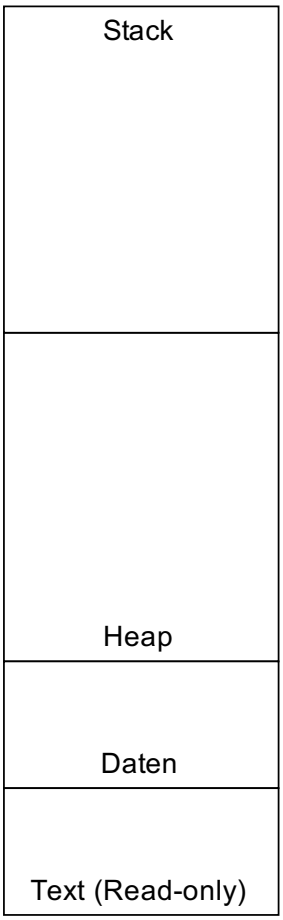
Lebensspanne **Sichtbarkeit**

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 **markierten Variablen (`x,pi,z,op1,c,a,b,m_1,m_2,d`)** dem Segment zu, in welchem sie durch den Compiler bzw. die Laufzeitumgebung abgelegt werden (Im Fall von Zeigervariablen ist der Zielort des Zeigers gemeint). Geben sie außerdem an, wo der eigentliche **Programmcode** von **`myAdd()`** 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 Programm führt bei Ausführung zu einem Adressbereichsfehler (*segmentation fault*). Beschreiben Sie kurz das Problem dieser Implementierung und korrigieren Sie den Code.

```
#include <iostream>
#include <vector>
// Beheben Sie das Problem in diesem Code
class MyClass {
  public:
    MyClass(int value) : data(value) {}
    char data;
};
MyClass* myFunction(char val) {
    MyClass myObject(val);
    return &myObject;
}
int main() {
    char userInput;
    std::vector<MyClass*> myClassObjects;
    std::cout << "Press 'y' to continue: ";
    do {
      std::cin >> userInput;
      myClassObjects.push_back(myFunction(userInput));
    } while (userInput != 'y');
    for (const auto& obj : myClassObjects) {
      std::cout << obj->data;
      delete obj;
    }
    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   = 7;
    int *pa = &a;
    int b   = *pa - 3;
    cout << "A) " << a << ", " << b << ", " << *pa << endl;

    int c   = 13;
    int d   = 7;
    int *pc = &c;
    int *pd = &d;
    *pc     = 5;
    d      -= 3;
    c      += 3;
    *pc     = *pd;
    cout << "B) " << c << ", " << d << ", " << *pc << ", " << *pd;
    cout << endl;

    short e   = 5;
    short f[] = {11, 13, 15, 17};
    *(f+2)    = *f + e;
    e         = f[1];
    ++(*f);
    cout << "C) " << f[0] << ", " << f[1] << ", " << f[2] << ", "
                  << f[3] << ", " << e    << endl;

    long g[] = {30, 20, 15, 10};
    long *p  = &g[2] + 1;
    *(p-2)   = *p + 1;
    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:

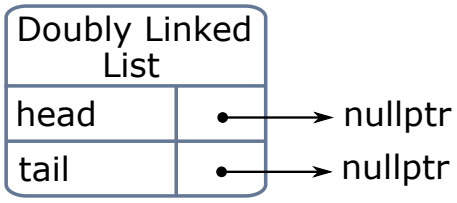
b) *[6 Punkte]* Die Klasse `DoublyLinkedList` repräsentiert eine doppelt verkettete Liste, die Werte vom Typ `int` speichert. Jedes Listenelement enthält einen Zeiger auf den Vorgänger und den Nachfolger. Sie ist wie folgt definiert:

```
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 das erste Element
    Element *tail;              // Zeiger auf das letzte Element

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

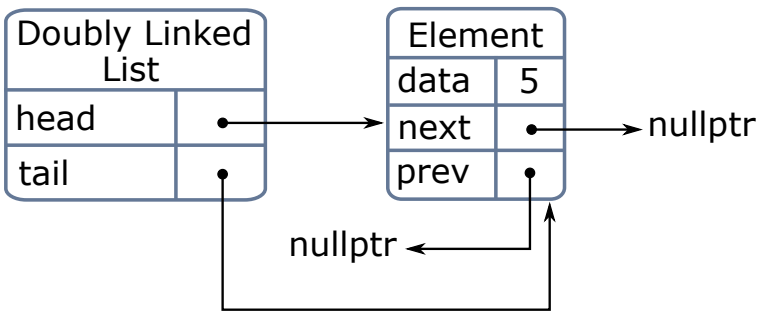
Ihre Aufgabe:
Implementieren Sie die Methode `void DoublyLinkedList::addFirst(int data)`, die ein Element am Anfang der Liste einfügt. Diese Methode muss in der Lage sein, Elemente in eine leere Liste und in eine bereits mit Elementen gefüllte Liste einzufügen. Verwenden Sie zur Bearbeitung dieser Aufgabe den vorgegebenen Coderahmen nach den Beispielen.

Beispiel:
Die nachfolgenden Abbildung veranschaulicht die Funktionsweise einer doppelt verketteten Liste.

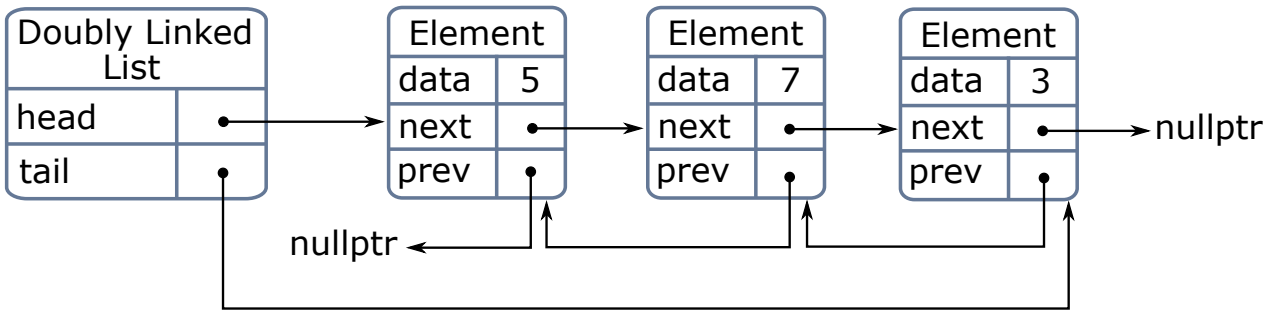
Doppelt verkettete leere Liste:



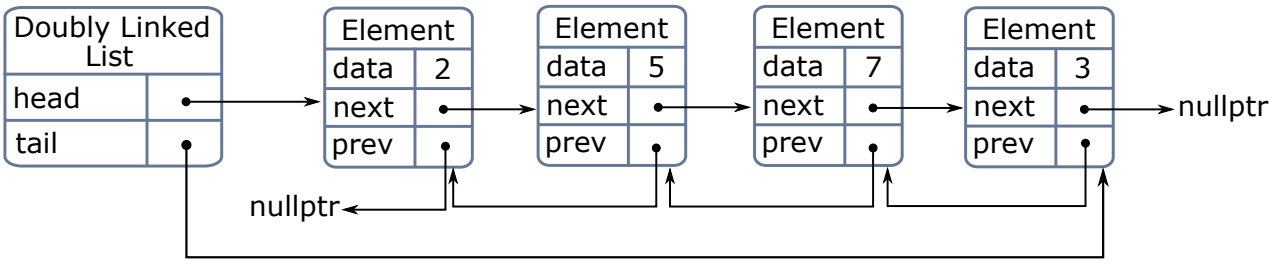
Doppelt verkettete Liste mit einem Element:



Doppelt verkettete Liste mit drei Elementen:



Nach dem Aufruf von `addFirst(2)` sieht die Beispiel-Liste wie folgt aus:



```
void DoublyLinkedList::addFirst(int data) {
  // Fuegen Sie hier Ihren Code ein









} // Ende von DoublyLinkedList::addFirst(int data)
```

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

a) *[4 Punkte]*
Eine Bank bietet das Sparprogramm *Kapital789* an. Dabei zahlt man einmal 789 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 drei Prozent des Guthabens fällig, wenn das Guthaben 1800 Euro übersteigt. Für das Guthaben K_n nach n Jahren ergibt sich damit

$$K_n = \begin{cases} 789 & \text{, wenn } n = 0, \\ (1+p) \cdot K_{n-1} - \frac{3 \cdot K_{n-1}}{100} & \text{, wenn } n > 0 \text{ und } K_{n-1} > 1800 \\ (1+p) \cdot K_{n-1} & \text{, sonst.} \end{cases}$$

Implementieren Sie eine Funktion `double guthaben(unsigned int n, double p)` in C++, die für gegebenes p rekursiv das Guthaben nach n Jahren nach obiger Vorschrift mit möglichst wenigen Schritten berechnet und zurückgibt. Das übergebene p muss nicht auf Gültigkeit überprüft werden.

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









} // Ende von guthaben()
```

b) *[6 Punkte]*
Gegeben sei die Signatur der Funktion `ausgabe`, der ein Feld `array` von ganzen Zahlen und dessen Länge `n` übergeben wird:

```
int ausgabe(int *array, unsigned int n)
```

Implementieren Sie `ausgabe` als rekursive Funktion, so dass die Elemente aus dem Feld von außen nach innen und in gegebener Reihenfolge ausgegeben werden. Das heißt, dass zuerst die erste und letzte Zahl, dann die zweite und vorletzte Zahl, usw., auf der Konsole mit `cout` ausgegeben wird. Zusätzlich soll die Gesamtsumme aller Elemente zurückgegeben werden.

Beispiel: Für das Feld `{1, -3, 5, -4, 2}` soll die Funktion die Ausgabe `1 2 -3 -4 5` erzeugen und den Wert `1` zurückgeben.

Wichtig: Die Schlüsselwörter **`for`**, **`while`**, **`goto`**, **`static`** sowie globale Variablen dürfen nicht benutzt werden.

(*Hinweis*: Es ist nicht notwendig, aber Sie dürfen die Signatur der Funktion verändern)

```
// Ihre Implementierung der Funktion ausgabe:
```

Aufgabe 5: *Klassen* [10 Punkte]

Hinweise für alle Teilaufgaben: In dieser Aufgabe sollen Sie Roboter, die mit einer künstlichen Intelligenz ausgestattet sein können, modellieren. Wählen Sie für Attribute und Methoden geeignete Modifikatoren für die Sichtbarkeit. Nutzen Sie die Möglichkeiten der Vererbung an geeigneter Stelle.

a) *[2 Punkte]*
Definieren Sie eine Klasse `KI`, die ein privates Attribut Intelligenz vom Typ `int` besitzt. Implementieren Sie einen öffentlichen parameterlosen Konstruktor, der `intelligenz` auf `150` setzt, sowie öffentliche Methoden zum Abfragen und Setzen des Attributs. Eine KI gilt als *smart*, genau dann wenn ihr Intelligenzwert mindestens 100 ist. Schreiben Sie eine öffentliche parameterlose Methode `istSmart`, die als `bool` zurückgibt, ob die KI *smart* ist.

```
// Ihre Implementierung der Klasse KI:
```

b) *[1 Punkt]*
Benötigt Ihre Klasse `KI` einen Destruktor? Wenn ja, dann implementieren Sie diesen; wenn nein, dann geben Sie eine Begründung an.

```
// Ihre Implementierung des Destruktors oder Begruendung:
```

In den folgenden Aufgaben c)–e) sollen die Klassen `Arbeiter`, `Roboter` und `SmartRoboter` implementiert werden. Beachten Sie, dass `SmartRoboter` von `Roboter` abgeleitet wird, und `Roboter` von der abstrakten Klasse `Arbeiter` abgeleitet wird.

c) *[3 Punkte]*
Implementieren Sie die abstrakte Klasse `Arbeiter`, welche ein privates `double`-Attribut `kraft` besitzt. Die Kraft soll durch den Konstruktor gesetzt werden und die öffentliche Methode `getKraft` soll den Wert zurückgeben. Zusätzlich soll die rein virtuelle öffentliche Methode `work` deklariert werden. Diese Methode hat keine Parameter und gibt die erbrachte Leistung als `double` wieder.

```
// Ihre Implementierung der Klasse Arbeiter:
```

d) *[2 Punkte]*
Leiten Sie die Klasse `Roboter` von der abstrakten Klasse `Arbeiter` ab. Die Klasse `Roboter` soll zusätzlich ein privates `bool`'sches Attribut `sparen` bekommen, das angibt, ob der Roboter sich im Energiesparmodus befindet. Implementieren Sie einen öffentlichen Konstruktor mit einem Parameter für den Energiesparmodus. Der Konsruktor soll `sparen` durch den Parameter initialisieren, und `kraft` mittels Basisklassenkonstruktor auf 100 setzen.

Überschreiben Sie die Methode `work`, die die erbrachte Leistung zurückgibt. Falls der Roboter nicht im Energiesparmodus ist, entspricht die Leistung der Kraft. Im Energiesparmodus entspricht die erbrachte Leistung 80% der normalen Leistung.

```
// Ihre Implementierung der Klasse Roboter:
```

e) *[2 Punkte]*
Die Klasse `SmartRoboter` wird von `Roboter` abgeleitet und hat eine `KI`. Hierfür soll die Klasse als privates Attribut einen Zeiger auf eine `KI` besitzen. Eine öffentliche Methode `entferneKI` ersetzt den Zeiger durch einen `nullptr`.

Ein öffentlicher Konstruktor erhält einen `bool`-Wert für den Energiesparmodus und einen Zeiger auf eine `KI` als Parameter, und soll die entsprechenden Attribute mit den übergebenen Werten initialisieren. Die Kraft des SmartRoboters soll 100 sein.

Die Methode `work` soll wieder die erbrachte Leistung berechnen. Diese errechnet sich aus dem Ergebnis der `work`-Methode der Basisklasse `Roboter`, multipliziert mit einem Effizienzfaktor. Der Effizienzfaktor ist 1, falls der `SmartRoboter` keine KI besitzt, 2, falls der `SmartRoboter` eine *nicht-smarte* `KI` besitzt und 3, falls der SmartRoboter eine *smarte* `KI` besitzt.

```
// Ihre Implementierung der Klasse SmartRoboter:
```

Aufgabe 6: ***Schablonen und Ausnahmen*** [10 Punkte]
a_1) Klassen-Schablone *[3,5 Punkte]*
Im folgenden Programm ist eine Klasse zur Verwaltung von `n` Elementen vom Typ `double` gegeben. Neben denen in der Klasse enthaltenen Methoden zur Manipulation der Sammlung gibt es noch zwei Funktionen zur Addition zweier Elemente sowie zur Ausgabe.

```
class Sammlung {
private:
    int n;
    double *elemente;
public:
    Sammlung(int _n): n(_n) {
        elemente = new double[n];
    }
    int size() {  return n; }
    double gib(int i) {     return elemente[i]; }
    void setze(int i, double wert) {
        elemente[i] = wert;
    }
    ~Sammlung() {
        delete[] elemente;
    }
};

double addiere(Sammlung p1, Sammlung p2, int pos) {
    return p1.gib(pos) + p2.gib(pos);
}

void print(Sammlung p) {
    for (int i = 0; i < p.size(); i++) {
        cout << i << ": " << p.gib(i) << endl;
    }
}
```

Ändern Sie die Klasse `Sammlung` zu einer Klassenschablone, sodass eine Sammlung, wie vorher, eine beliebige Anzahl an Elementen desselben, aber beliebigen Typs enthalten kann. Ändern Sie außerdem die beiden Funktionen `addiere` und `print` in eine Funktionsschablone. Gehen Sie davon aus, dass die nötigen Header-Dateien eingebunden wurden und Ihnen der Namensraum `std` zur Verfügung steht.

```
// 6a1) Modifizieren sie den Code mittels einfuegen und durchstreichen
class Sammlung {
    private:
    int n;
    double *elemente;
    public:
    Sammlung(int _n): n(_n) {
        elemente = new double[n];
    }
    int size() {  return n; }
    double gib(int i) {     return elemente[i]; }
    void setze(int i, double wert) {
        elemente[i] = wert;
    }
    ~Sammlung() {
        delete[] elemente;
    }
};

double addiere(Sammlung p1, Sammlung p2, int pos) {
    return p1.gib(pos) + p2.gib(pos);
}

void print(Sammlung p) {
    for (int i = 0; i < p.size(); i++) {
        cout << i << ": " << p.gib(i) << endl;
    }
}
```

a_2) Verwendung der Klassen-Schablone *[2 Punkte]*
Ergänzen Sie eine `main`-Funktion, sodass zwei Instanzen von der Klasse `Sammlung` mit Elementen des Typs `double` angelegt werden. Beide Instanzen sollen die Größe zwei haben. Danach soll mit Hilfe der Funktion `setze` die Elemente an Stelle 0 von beiden Instanzen einen beliebigen Wert zugewiesen bekommen. Nun sollen diese mit der Funktion `addiere` aufsummiert und mittels `cout` ausgegeben werden.

```
// 6a2) Schreiben sie die Methode main
int main() {









    return 0;
}
```

b) Ausnahmebehandlung *[4,5 Punkte]*
Betrachten Sie das nachfolgend angegebene Programm. Es nutzt die Methode `doSomething` der Klasse **Eval**. Die Implementierung dieser Methode ist Ihnen unbekannt und es besteht die Möglichkeit, dass sie bei ungültigen Eingaben eine Exception wirft. Zudem arbeitet das angegebene Programm mit Fehlercodes, welche durch `err` übergeben werden. Verändern Sie die Funktion und das Hauptprogramm, indem Sie die Fehlerbehandlung mit Fehlercodes durch eine Ausnahmebehandlung mit *C++ Exceptions* ersetzen. Nutzen Sie dazu eine sinnvolle Anzahl an Klassen zur differenzierten Fehlerbehandlung. Nutzen Sie dabei **keine** Vererbung. Stellen Sie **zudem** sicher, dass jegliche von der Methode `doSomething` geworfenen Exceptions in der `main` Funktion gefangen werden.

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

class Eval {
private:
    double x, y;
public:
    Eval(double x, double y) : x(x), y(y) {}
    double doSomething() {
        if (x + y < 1) throw "Error";
        return x + y;
    }
};

double f(double x, double y, unsigned int &err) {
    if (x < 0 && y < 0) {
        err = 1;
        return 0;
    }
    if (x > 1000 || y > 1000) {
        err = 2;
        return 0;
    }
    Eval e(x, y);
    return e.doSomething();
}

int main() {
    unsigned int err = 0;
    double result = f(-0.3, -0.3, err);
    if (err == 0)
        cout << result << endl;
    else if (err == 1)
        cout << "Fehler 1" << endl;
    else if (err == 2)
        cout << "Fehler 2" << endl;

    return 0;
}
```

```
// 6b) Ergaenzen sie den Code
#include <iostream>
using namespace std;









class Eval {
    private:
        double x, y;
    public:
        Eval(double x, double y) : x(x), y(y) {}
        double doSomething();  // Sollen Sie nicht implementieren
};

double f(double x, double y) {




```

```
}
int main() {

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