Technische Universität Dortmund

Klausur „Einführung in die Programmierung“

	erreichbare Punkte	erhaltene Punkte			
Aufgabe 1	20	**a**	**b**	**c**	
Aufgabe 2	20	**a**	**b**		
Aufgabe 3	20	**a**	**b**		
Aufgabe 4	20	**a**	**b**		
Aufgabe 5	20	**a**	**b**	**c**	
Aufgabe 6	20	**a**	**b**		
Summe	120				
Spickzettel vorhanden?					

(Matrikel-Nr.)

(Name) (Vorname)

Durch meine Unterschrift bestätige ich

- den Empfang der vollständigen Klausur (22 Seiten inklusive Deckblatt),
- die Kenntnisnahme der Hinweise auf Seite 2.

Dortmund, 05.02.2025

(Unterschrift)

Hinweise

Bitte lesen Sie die folgenden Informationen **aufmerksam** und unterschreiben Sie die Erklärung auf der ersten Seite.

- Es können **120 Punkte** erreicht werden; 50% der Gesamtpunktzahl sind zum Bestehen der Klausur erforderlich. Zur Bearbeitung stehen insgesamt **180 Minuten** zur Verfügung.
- Als Hilfsmittel ist lediglich ein **von Ihnen eigenhändig geschriebenes** (Handschrift, kein Ausdruck oder Kopie) **A4-Blatt** (beidseitig), welches eingesammelt wird. Ansonsten dürfen **keine** weiteren Hilfsmittel verwendet werden.
- Die Antworten sind in **deutscher** oder **englischer** Sprache zu verfassen.
- Notieren Sie Ihre Lösungen direkt auf den ausgeteilten Aufgabenblättern unter Verwendung eines dokumentenechten, schwarzen oder blauen Stifts. Entfernen Sie **nicht** die Heftung der Blätter. Falls der vorgesehene Platz nicht ausreichen sollte, können Sie auch die Rückseiten der Blätter und die Reserveseiten am Ende verwenden. Notieren Sie in diesem Fall aber an der für die Lösung vorgesehenen Stelle einen Verweis auf die Seite.
- Da es unterschiedliche Sprachgebräuche sowie Algorithmen- und Konzept-Ausprägungen gibt, werden hier ausdrücklich die aus Vorlesung und Übungen bekannten Ausdrucksweisen und Ausprägungen zu Grunde gelegt.

Aufgabe 1: Programmverständnis (20 Punkte)

a) Kontrollstrukturen

6 Punkte

Geben Sie die Ausgabe des folgenden C++-Programms an. Das Symbol ␣ steht für ein Leerzeichen.

```
1  #include <iostream>
2  using namespace std;
3
4
5  void f(int i) {
6    i = i - 2;
7  }
8
9  int main(){
10   int i = 7, k = 12;
11
12   do {
13     k -= 2;
14     switch (i) {
15       case 6:
16       case 1:
17         cout << i + k;
18         break;
19       case 2:
20         cout << i;
21       case 5:
22         cout << i + 1;
23         break;
24       default:
25         cout << k + 3 * i;
26     }
27
28     f(k);
29     --i;
30     cout << ",";
31   } while (k > 0);
32   cout << endl;
33
34   return 0;
35 }
```

Ausgabe:

---

b) Parameterübergabe

9 Punkte

Geben Sie die Ausgabe des folgenden C++-Programms an. Das Symbol ␣ steht für ein Leerzeichen.

```
1  #include <iostream>
2  using namespace std;
3
4  void f(int k) {
5    for (int i = 0; i < 5; i++) {
6      k = k - 1;
7    }
8  }
9
10 int g(int *i, int a) {
11   *i -= 1;
12   if (a == 0) {
13     return *i;
14   } else {
15     return *i + g(i, a - 1);
16   }
17 }
18
19 void h(int &k) {
20   for (int i = 0; i < 4; i++) {
21     k = k * 2;
22   }
23 }
24
25 int main(){
26   int k = 5;
27
28   f(k);
29   cout << k << endl;
30
31   cout << g(&k, 3) << endl;
32
33   h(k);
34   cout << k << endl;
35
36   return 0;
37 }
```

Ausgabe:

---

c) Fehlersuche

5 Punkte

In dem folgenden Programm haben sich mehrere Fehler versteckt. Sie können zu einem Fehler beim Übersetzen *oder* zu einem potentiellen Fehler während der Ausführung führen. Pro Zeile versteckt sich maximal *ein* Fehler. Das Symbol ␣ steht für ein Leerzeichen.

Nennen Sie pro Fehler die Zeile, in der er vorkommt, und was genau das Problem ist.

```
1  #include <iostream
2  using std::cout;
3
4
5  int main() {
6    int k[] = {1, 6, 4, 3, 5, 2};
7    int result = 0
8
9    for (int i = 0;  i < 7; i++) {
10     resutl += k[i];
11   }
12
13   cout << "result␣=␣" << result << endl;
14
15   return 0;
16 }
```

Ausgabe:

Aufgabe 2: Speicherverwaltung (20 Punkte)

a) Speichersegmente

13 Punkte

Markieren Sie alle Bezeichner in dem folgenden C++-Code und tragen Sie diese in die unten stehende Liste der Speichersegmente ein. Ordnen Sie ebenfalls den Inhalt von `*r` einem Segment zu.

```
1  #include <iostream>
2  #define abs(x) ((x) < 0 ? -(x) : (x))
3
4  constexpr double phi = 1.6180339887;
5  constexpr int iter = 42;
6  int counter = 0;
7
8  double calc(double a, double b, int d) {
9      ++counter;
10     if (d == 0) { return b / a; }
11     return calc(b, a + b, d - 1);
12 }
13
14 void print(double z) {
15     std::cout << "|z␣-␣phi|␣=␣" << abs(z - phi) << std::endl;
16 }
17
18 int main() {
19     double x = 1.0;
20     double y = 1.0;
21     double* r = new double;
22     *r = calc(x, y, iter);
23     print(*r);
24     delete r;
25     return 0;
26 }
```

Stack:

Heap:

BSS-Segment:

Datensegment:

read-only Datensegment:

Textsegment:

b) Adressberechnung

7 Punkte

In diese Aufgaben sollen Sie den Abstand zwischen zwei Arrayelementen eines *zweidimensionalen* Arrays bestimmen. Beachten Sie hierbei die zwei unterschiedlichen Arten zur Bestimmung der Differenz in den Zeilen 7-9. Gehen Sie davon aus, dass der Datentyp `float` eine Größe von 32bit hat.

Der Operator `reinterpret_cast⟨T*⟩(ptr)` wird in C++verwendet, um den Parameter `ptr` in einen beliebigen anderen Datentyp `T*` umzuwandeln. Es findet hierbei *keine* Typüberprüfung statt. Die darunterliegenden Bitmuster im Speicher werden nun als der Zieldatentyp interpretiert.

```
1  #include <iostream>
2  using namespace std;
3
4  float block[5][2];
5
6  int main() {
7      cout << &block[3][1] - &block[0][0] << '|';
8      cout << reinterpret_cast<char*>(&block[3][1])
9           - reinterpret_cast<char*>(&block[0][0]) << endl;
10     return 0;
11 }
```

Ausgabe:

(Wenn Sie sich bei einem der beiden Ausgaben unsicher sind, schreiben Sie nur einen Teil hin.)

Wie lautet die absolute Adresse des Elements `block[3][1]`, wenn das Element `block[0][0]` die Adresse 38 hat?

Aufgabe 3: Zeiger (20 Punkte)

a) Zeigerarithmetik

8 Punkte

Gegeben sei folgendes Programm, welches ausführlich Gebrauch von Zeigern macht.

```
1  #include <iostream>
2  using namespace std;
3
4  const int ONE = 1;
5
6  int main() {
7      int a = 4;
8      int b = a + 4;
9      int* pa = &a;
10     int c = *pa + 11;
11
12     cout << "A) " << a << ", " << b << ", " << c << endl;
13
14     int d = 15 + *(&ONE);
15     int* pd = &d;
16     int* p = pa + 4;
17     int e = *(p - a) + 19;
18
19     cout << "B) " << *pd << ", " << e << endl;
20
21     short f[] = {42, 8, 100};
22     short* g = &f[-1];
23     *(f+1) += g[3];
24
25     cout << "C) " << g[1] << ", " << g[2] << endl;
26
27     char h[] = "OBOTE";
28     char* i = &h[0];
29     h[0] = h[1];
30     h[1] = *i;
31
32     cout << "D) " << h << endl;
33
34     return 0;
35 }
```

Welche Ausgabe erzeugt das gegebene Programm?

Platz für Notizen:

b) Verkettung

12 Punkte

Gegeben Sei folgende Header-Datei zur Klasse `Book`, die ein Buch mit Seiten umsetzt. Dabei besitzt jede `Page` (Seite) einen Inhalt, eine Referenz auf die Seite vor ihr und eine Referenz auf die Seite nach ihr. Das Buch selbst hält mit Referenzen nach, was seine erste bzw. letzte Seite ist und zählt, wie viele Seiten es insgesamt enthält.

Implementieren Sie die Funktion `tearOutPage`, die das herausreißen einer Seite aus dem Buch simuliert. Die Funktion nimmt als Parameter die Seitenzahl des Buches (`pageNumber`), die herausgerissen werden soll entgegen. Außerdem soll ein Zeiger auf die herausgerissene Seite zurückgeben werden.

Seitenzahlen in Büchern beginnen bei 1. Falls die Seite mit der gewünschten Seitenzahl nicht im Buch enthalten sein sollte, soll stattdessen ein `NULL`-Zeiger zurückgegeben werden. Es wird *niemals* die erste oder die letzte Seite aus einem Buch gerissen werden.

```
1  #include <iostream>
2  using namespace std;
3
4  class Book{
5  private:
6      struct Page{
7          string pageContent; // Seiteninhalt
8          Page* previousPage; // Zeiger auf die vorherige Seite
9          Page* nextPage;     // Zeiger auf die nachfolgende Seite
10     };
11     Page* firstPage;
12     Page* lastPage;
13     int numberOfPages;
14
15 public:
16     Book() {
17         firstPage = lastPage = nullptr;
18         numberOfPages = 0;
19     }
20     ~Book() { destroy(); }
21     void destroy();
22     int getNumberOfPages() { return this->numberOfPages; }
23     Page* tearOutPage(int pageNumber); // Zu implementieren
24 };
```

```
1 #include "a3-b.hpp"

Book::Page* Book::tearOutPage(int pageNumber) {

}
```

Aufgabe 4: Rekursion (20 Punkte)

a) Binärer Suchbaum

12 Punkte

Gegeben sei die Implementierung eines binären Suchbaums, deren relevante Bestandteile unten dargestellt sind.
Implementieren Sie die private Methode `evenLevelSum`, die rekursiv die Summe der Werte (`data`) aller Knoten berechnet, deren Tiefe eine *grade* Zahl ist.
Die Tiefe eines Knotens gibt die Anzahl der Kanten bis zur Wurzel an. Die Wurzel hat Tiefe 0 und steigt mit jeder weiteren Kante nach unten um 1 an – siehe Zeile 18. Ein leerer Baum hat keine Knoten.

```
1  #include <iostream>
2  using namespace std;
3
4  class BinTree {
5  private:
6    struct Node {
7      int data;
8      Node *left, *right;
9    } *root;
10   Node *insert(Node *node, int value);
11   int evenLevelSum(Node *node, int depth);
12 public:
13   BinTree() : root(nullptr) { }
14   void insert(int value) {
15     root = insert(root, value);
16   }
17   int evenLevelSum()  {
18     return evenLevelSum(root, 0);
19   }
20   ~BinTree();
21 };
```

```
int BinTree::evenLevelSum(Node *node, int depth) {
```

```
}
```

b) Sparprogramm

8 Punkte

Eine Bank bietet das Sparprogramm *Kapital500* an. Dabei zahlt man ein Mal 500 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 ein Prozent des Guthabens fällig, wenn das Guthaben 2000 Euro übersteigt. Für das Guthaben nach n Jahren ergibt sich damit

$$K_n = \begin{cases} 500 & \text{, wenn } n = 0, \\ (1+p) \cdot K_{n-1} - \frac{K_{n-1}}{100} & \text{, wenn } n > 0 \text{ und } K_{n-1} > 2000 \\ (1+p) \cdot K_{n-1} & \text{, sonst.} \end{cases}$$

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

```
double guthaben(unsigned int n, double p) {
}
```

Aufgabe 5: Klassen (20 Punkte)

In dieser Aufgaben sollen Sie eine Klassenhierarchie mit virtuellen Methoden realisieren.
Sie sollen die *abstrakte* Klasse `TUMember`, die ein(e) Angehörig(e) der TU-Dortmund repräsentiert, implementieren. Sofern es *nicht* anders angegeben wird, soll alles öffentlich sichtbar sein. Von der Oberklasse sollen Sie die beiden, konkreten Unterklassen `Student` sowie `Employee` ableiten. Die Funktionalität der Klassen sieht wie folgt aus:

a) Klasse `TUMember`

- Für jedes Mitglied der TU werden die Informationen Vorname, Nachname und E-Mail-Adresse als Attribute verwaltet. Sie sollen nur für sich selbst sowie für abgeleitete Klassen sichtbar sein.
- Dem Konstruktor werden drei Parameter, die den Attributen entsprechen, übergeben und die Attribute dementsprechend initilisiert.
- Für die Attribute Vor- und Nachname soll es jeweils eine Methode geben, die das Attribut zurückliefert.
- Für das Attribut E-Mail-Adresse soll jeweile eine Methode zum Abfragen und zum Setzen geben.
- Ferner soll jedes TU-Mitglied zwei *rein virtuelle* Methoden `toString()` sowie `getId()` besitzen. Die Methode `toString()` liefert einen String und die Methode `getId()` liefert einen Integer zurück.

b) Klasse `Student`

- Jede Student:in hat *zusätzlich* noch ein eigenes Attribut Matrikelnummer, das nur ihr gehört.
- Der Konstruktor für eine Student:in bekommt alle *vier* Attribute übergeben. Das eigene Attribut soll entsprechend gesetzt werden. Außerdem sollen Vorname, Nachname und E-Mail-Adresse mit Hilfe des Konstruktors der Oberklasse gesetzt werden.
- Eine Student:in implementiert die Methode `getId()`. Sie gibt hierbei die Matrikelnummer zurück.
- Die Methode `toString()` soll eine Zeichenkette der Form „Student: XXX“ zurückgeben. Wobei der Platzhalter `XXX` durch den jeweiligen Nachnamen ersetzt werden soll.

c) Klasse `Employee`

- Jede Mitarbeiter:in hat *zusätzlich* noch ein eigenes Attribut Mitarbeiternummer, das nur ihr gehört.
- Der Konstruktor für eine Mitarbeiter:in bekommt alle *vier* Attribute übergeben. Das eigene Attribut soll entsprechend gesetzt werden. Außerdem sollen Vorname, Nachname und E-Mail-Adresse mit Hilfe des Konstruktors der Oberklasse gesetzt werden.
- Eine Mitarbeiter:in implementiert die Methode `getId()`. Sie gibt hierbei die Mitarbeiternummer zurück.
- Die Methode `toString()` soll eine Zeichenkette der Form „Employee: XXX“ zurückgeben. Wobei der Platzhalter `XXX` durch den jeweiligen Vornamen ersetzt werden soll.

```
1 #include <iostream>
2 #include <deque>
3 using namespace std;
```

a) Klasse `TUMember` (12 Punkte)

```
class TUMember {
};
```

b) Klasse `Student` (4 Punkte)

```
class Student : public TUMember {

};
```

c) Klasse Employee (4 Punkte)

```
class Employee : public TUMember {

};
```

Aufgabe 6: Schablonen & Ausnahmen (20 Punkte)

a) Summentyp als Schablone (9 Punkte)

Die unten angegebene Klasse stellt eine Summentypen bereit. Mit dessen Hilfe können Werte verschiedener Typen als ein gemeinsamer Typ verpackt und später wieder extrahiert werden.

In dieser konkreten Implementierung können Werte entweder vom Typ `int` oder `string` sein.

Ändern Sie die Klasse zu einer Klassen-Schablone mit zwei Typ-Parametern (Vorzugsweise als A und B benannt), die Werte von beliebigen Typen halten kann. Ändern Sie außerdem die `main`-Funktion so ab, dass Sie die Schablone benutzt.

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**.

```
1 #include <iostream>
2 #include <string>
3 using namespace std;

class Summentyp {

public:

  Summentyp() : value(nullptr), type(0) {}

  Summentyp(int value) : value(new int(value)), type('A') {}

  Summentyp(string value) : value(new string(value)), type('B') {}

  int *getValueA() const {

    if (type == 'A') {

      return static_cast<int *>(value);

    } else { return nullptr;}
  }
```

```
  string *getValueB() const {
    if (type == 'B') {
      return static_cast<string *>(value);
    } else { return nullptr;}
  }
  void setValue(int value) {
    deleteValue();
    this->value = new int(value);
    type = 'A';
  }
  void setValue(string value) {
    deleteValue();
    this->value = new string(value);
    type = 'B';
  }
  char getVariante() { return type; }
```

```
  void deleteValue() {

    switch (type) {

    case 'A':

      delete static_cast<int *>(value);

      break;
    case 'B':

      delete static_cast<string *>(value);

    default:
      break;
    }
    value = nullptr;
    type = 0;
  }

  ~Summentyp() { deleteValue(); }

private:

  void *value;
  char type;
};
int main() {

  Summentyp summentyp;

  summentyp.setValue("Hallo");
  cout << "Wert: " << *summentyp.getValueA() << endl;
  return 0;
}
```

b) Ausnahmen (11 Punkte)

Das nachfolgend angegebene Programm arbeitet mit Fehlercodes. Verändern Sie die Funktion und das Hauptprogramm, indem Sie die Fehlerbehandlung mit Fehlercodes ersetzen durch eine Ausnahmebehandlung mit C++-Exceptions. Nutzen Sie dazu eine sinnvolle Anzahl an Klassen zur differenzierten Fehlerbehandlung. Der Parameter `ungueltigerWert` soll nicht mehr verwendet werden. Notieren Sie Ihre Lösung **auf den nächsten Seiten**.

Die Bedeutung der Fehlercodes sowie das daraus resultierende Verhalten in der ursprünglichen Implementierung sieht wie folgt aus:

- **0:** Es trat kein Fehler auf.
- **1:** Es wurde ein leeres Array übergeben. Es erfolgte keine Berechnung.
- **2:** Der im Array gespeicherte Wert liegt außerhalb des Wertebereichs. In diesem Fall wird der ungültige Wert in der Referenz `ungueltigerWert` abgelegt.

```
1  #include <iostream>
2  using namespace std;
3
4  int berechneWerte(int * werte, unsigned int const n,
5          int &ungueltigerWert) {
6      int errCode = 0;
7      if (n == 0) return errCode = 1;
8      for (unsigned int i = 0; i < n; ++i) {
9          if (werte[i] < 0 || werte[i] > 2047) {
10             errCode = 2;
11             ungueltigerWert = werte[i];
12         }
13         werte[i] *= (werte[i] + 1);
14     }
15     return errCode;
16 }
17
18 int main() {
19     int const n = 4;
20     int werte[n] = { 2, 1, -2, 1 };
21     int ungueltigerWert = 0;
22     int errCode = berechneWerte(werte, n, ungueltigerWert);
23     if (errCode == 1)
24         cerr << "Fehler:␣Arraygroesse␣0" << endl;
25     else if (errCode == 2)
26         cerr << "Fehler:␣Array␣enthaelt␣ungueltigen␣Wert␣"
27              << ungueltigerWert << endl;
28     else {
29         for (unsigned int i = 0; i < n; ++i) {
30             cout << werte[i] << endl;
31         }
32     }
33     return 0;
34 }
```

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

```
int main() {
  int const n = 4;
  int werte[n] = { 2, 1, 2, 0 };
}
```