Fakultät IV
NI/CV

Einführung in die Informatik (C/C++) - Probeklausur

10.02.2020 - 14.02.2020

Musterlösung
Stand: 6. Februar 2020

Aufgabe	Punkte	Erreichte Punkte
Programmieren in C/C++		
1	10	
2	10	
3	12	
4	9	
5	9	
6	10	
Rechneraufbau		
7	10	
8	6	
9	7	
10	10	
11	7	
Summe	100	

Aufgabe 1 (10 Punkte) Allgemeine Fragen zu C und C++.

In dieser Aufgabe ist jeweils genau eine Antwort richtig, welche Sie ankreuzen sollen. Kreuzen Sie pro Teilaufgabe nur ein Kästchen an. Eine richtige Antwort ergibt einen Punkt, eine falsche 0 Punkte. Es gibt keine Minuspunkte. Um ein versehentlich gesetztes Kreuz wieder zu löschen, füllen Sie das jeweilige Kästchen aus und zeichnen ein leeres daneben.

1. (1 Punkt) Welcher Datentyp steht in C für Ganzzahlen zur Verfügung?

   ☐ **string**
   ☒ **int**
   ☐ **bool**
   ☐ **double**

2. (1 Punkt) Welche Anweisung kann **nicht** benutzt werden, um eine Schleife zu programmieren?

   ☐ `while(bedingung) { /*code*/ }`
   ☐ `do { /*code*/ } while (bedingung);`
   ☒ `repeat { /*code*/ } until (bedingung);`
   ☐ `for(bedingung) { /*code*/ }`

3. (1 Punkt) Welches der folgenden Wörter ist **kein** Schlüsselwort in C++?

   ☐ **static**
   ☒ **primary**
   ☐ **try**
   ☐ **virtual**

4. (1 Punkt) Was ist der Unterschied, zwischen Heap- und Stackspeicher in C/C++?

   ☒ Stackspeicher wird automatisch verwaltet, Heapspeicher muss vom Programmierer angefordert und freigegeben werden
   ☐ Stackspeicher wird mit den Funktionen `malloc()` oder `calloc()` alloziert, Heapspeicher mit dem Schlüsselwort **new**
   ☐ Stackspeicher ist begrenzt, Heapspeicher ist unendlich
   ☐ Stackspeicher ist deutlich langsamer im Zugriff, daher sollte immer Heapspeicher verwendet werden

5. (1 Punkt) Es soll eine C++ Methode zum Vergleich von zwei Objekten implementiert werden, welche berechnet ob das eine Objekt *größer*, *kleiner* oder *genau so groß* ist, wie das andere. Welcher Rückgabetyp eignet sich für die Methode?

   ☒ **int**
   ☐ **bool**
   ☐ **void** A
   ☐ **bool***

6. (1 Punkt) Betrachten Sie den folgenden Codeausschnitt. Welchen Wert hat die Variable `d` am Ende?

```
1 int a = 9;
2 int b = 2;
3 double d = a / b + 1;
```

- ☐ 3
- ☒ 5
- ☐ 5.5
- ☐ 6

7. (1 Punkt) Was sind **`dynamic_cast`**s?

- ☐ Vergleichsmethoden
- ☐ Speicherreservierungen
- ☐ Exceptionhandler
- ☒ Typumwandlungen

8. (1 Punkt) Was trifft auf Rekursionen **nicht** zu?

- ☐ Sie rufen sich selbst erneut auf
- ☐ Sie gehen verschwenderischer mit Speicher um, als Iterationen
- ☒ Sie brauchen mindestens 2 verschiedene Rekursionsanker
- ☐ Sie könnten in der Regel Schleifen ersetzen

9. (1 Punkt) Welcher Anweisungsblock ist dazu geeignet, eine Exception aufzufangen?

- ☐ `/*code*/ catch (exception& e) { /*fehlerbehandlung*/ }`
- ☐ `try { /*code*/ } /*fehlerbehandlung*/`
- ☐ `try { /*code*/ catch (exception& e) { /*fehlerbehandlung*/ } }`
- ☒ `try { /*code*/ } catch (exception& e) { /*fehlerbehandlung*/ }`

10. (1 Punkt) Gegeben sind die folgenden beiden Klassen `A` und `B`:

```
1 class A { public: int a; };
2 class B : protected A { private: int b; };
```

Welche Sichtbarkeit hat das Attribut `a` bei einem Objekt der Klasse `B`?

- ☐ **`void`**
- ☐ **`private`**
- ☒ **`protected`**
- ☐ **`public`**

Aufgabe 2 (10 Punkte) Arrays in C.

Es soll eine C-Funktion geschrieben werden, die prüft, ob in einem übergebenen C-String irgendwo zwei oder mehr direkt aufeinanderfolgende Ziffern vorkommen (z.B. wie bei `"Ich habe 13 Euro."`). Um Ihnen diese Aufgabe zu erleichtern, soll sie in zwei Teilaufgaben erledigt werden.

1. (3 Punkte) Schreiben Sie eine C-Funktion **int** `istZiffer(`**char** `zeichen)`, die eine 1 zurückgibt, wenn es sich bei der übergebenen Variable um eine Ziffer handelt. Andernfalls soll eine 0 zurückgegeben werden.

   Hinweis: Eine Variable `zeichen` vom Typ **char** ist genau dann eine Ziffer, wenn `'0'` $\leq$ `zeichen` $\leq$ `'9'` gilt.

Listing 1: Lösung

```
1 int istZiffer(char zeichen)
2 {
3     if ( ('0' <= zeichen) && (zeichen <= '9')) {    // 3 Punkte:
4                     // if/return (1), Vergleiche (je 1)
5         return 1;                                   // 0 Punkte
6     } else {
7         return 0;                                   // 0 Punkte
8     }
9 }
```

Listing 2: Alternativ-Lösung

```
1 int istZiffer2(char zeichen)
2 {
3     return ('0' <= zeichen) && (zeichen <= '9');    // 3 Punkte:
4                     // if/return (1), Vergleiche (je 1)
5 }
```

2. (7 Punkte) Schreiben Sie eine C-Funktion **int** `ziffernHintereinander(`**char** `text[])`, die eine 1 zurückgibt, wenn in dem übergebenen C-String zwei oder mehr aufeinanderfolgende Ziffern vorkommen. Andernfalls soll auch hier eine 0 zurückgegeben werden. Verwenden Sie dabei die C-Funktion **int** `istZiffer(`**char** `zeichen)` aus dem vorherigen Aufgabenteil.

   Hinweis: Diese Teilaufgabe ist auch lösbar, wenn die vorherige Teilaufgabe nicht bearbeitet wurde.
   Beispielsweise ist die Rückgabe 1, wenn der übergebene C-String `"Ich habe 13 Euro."` ist.
   Eine Rückgabe von 0 käme z.B. vor, wenn `"Ich habe 2 oder 3 Euro."` übergeben wird.

Listing 3: Lösung

```
1  int ziffernHintereinander(char text[])
2  {
3      char temp = 0;                              // Hilfsvariable optional (0 Punkte)
4      for (int i = 0; text[i] != '\0'; ++i) { // Auch richtig: text[i+1] != '\0'
5          // 2 Punkte Schleife: Schleifenvariable init (0,5), update (0,5), Bedingung (1)
6          if (istZiffer(text[i]) && istZiffer(text[i+1])) {   // 3 Punkte:
7                                  // Aufrufe (je 1 Punkte), korrekte Bedingung (1 Punkt)
8              temp = 1;                           // 1 Punkt für gefunden
9          }
10     }
11     return temp;                                // 1 Punkt für nicht gefunden
12 }
```

Listing 4: Alternativ-Lösung

```
1 int ziffernHintereinander2(char text[])
2 {
3     while (*++text != '\0') {   // 2 Punkte: Pointer als Schleifenzeahler (2 bis n)
4         if (istZiffer(*text) && istZiffer(*(text-1))) { // 3 Punkte (s.o.)
5             return 1;                                   // 1 Punkt für gefunden
6         }   // Alternative if-Bedingung: (istZiffer(*--text) && istZiffer(*++text))
7     }
8     return 0;                                           // 1 Punkt für nicht gefunden
9 }
```

Aufgabe 3 (12 Punkte) Speicherverwaltung in C.

Gegeben sei das folgende, unvollständige C-Programm. Sie sollen nun die mit `// TODO` gekennzeichneten Stellen ergänzen. Schreiben Sie Ihren Programmcode dazu bitte immer unterhalb der jeweiligen Aufgabe, und nicht in die Vorgabe.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  static int array_len;
5
6  unsigned char * array_allocate(int len)
7  {
8      // TODO
9  }
10
11 /* Fügt Element num in arr[pos] ein. */
12 void array_insert(unsigned char *arr, unsigned char num, int pos)
13 {
14     if (pos >= 0 && pos < array_len) arr[pos] = num;
15 }
16
17 // TODO: array_get implementieren
18
19 int main(void)
20 {
21     // TODO: arr deklarieren und Speicher für 3 Werte allozieren
22
23     array_insert(arr, 13, 0);   // Wert 13 an Position 0 einfügen
24     array_insert(arr, 23, 1);   // Wert 23 an Position 1 einfügen
25     array_insert(arr, 42, 2);   // Wert 42 an Position 2 einfügen
26
27     for (int i=0; i < 3; ++i) printf("%d\n", array_get(arr, i));
28
29     // TODO: allozierten Speicher für arr freigeben
30 }
```

1. (6 Punkte) Implementieren Sie die Funktion `array_allocate`. Die Funktion soll Platz für `len` Elemente auf dem Heap allozieren und einen Pointer auf den allozierten Speicher zurückgeben. Falls kein Speicher alloziert werden kann, soll das Programm sofort beendet werden. Außerdem soll die globale Variable `array_len` den Wert von `len` erhalten.

   Hinweis: Zum sofortigen Beenden des Programms kann der Funktionsaufruf `exit(EXIT_FAILURE)` verwendet werden.

Listing 5: Lösung & Bewertung

```
1 unsigned char * array_allocate(int len)
2 {
3     array_len = len;     // 1 Punkt
4     unsigned char *arr = malloc(len * sizeof(unsigned char));    // Auch: malloc(len)
5     // 2,5 Punkte: malloc oder calloc (1), richtige größe (1), richtiger typ (0,5)
6     if (arr == NULL) exit(EXIT_FAILURE); // 1,5 Punkte: Abfrage (1), exit (0,5)
7     return arr; // 1 Punkt
8 }
```

2. (1 Punkt) Ergänzen Sie die Zeile 21 der Vorgabe. Wählen Sie einen geeigneten Datentyp für `arr`. Reservieren Sie unter Verwendung der Funktion `array_allocate` Speicher für 3 Elemente.

   Hinweis: Der Funktionskopf von `array_allocate` und `array_insert` könnte Ihnen bei der Wahl des Datentyps helfen.

Listing 6: Lösung & Bewertung

```
21     unsigned char *arr = array_allocate(3); // 0,5 Punkt Datentyp, 0,5 Punkte Aufruf
```

3. (4 Punkte) Deklarieren und definieren Sie die Funktion `array_get`, die in Zeile 27 der Vorgabe verwendet wird. Wählen Sie einen geeigneten Rückgabetyp und geeignete Funktionsparameter. Wenn auf Speicher außerhalb des allozierten Bereichs zugegriffen wird, soll -1 zurückgegeben werden.

   Hinweis 1: Orientieren Sie sich an der Funktion `array_insert`.
   Hinweis 2: Welcher Rückgabetyp ist groß genug um -1, als auch den Wertebereich von unsigned char zu umfassen? Zeile 27 der Vorgabe enthält einen Hinweis.

Listing 7: Lösung & Bewertung

```
1 int array_get(unsigned char *arr, int pos)  // Alternativer Rückgabetyp: short
2 {                                           // 1,5 Punkte: Rückgabewert (1), Rest (0,5)
3     if (pos >= 0 && pos < array_len) return arr[pos];   // 2 Punkte: Abfrage (1),
4                                                         // Rückgabewert (1)
5     return -1;  // 0,5 Punkte
6 }
```

4. (1 Punkt) Ergänzen Sie die Zeile 29 der Vorgabe, um den allozierten Speicher wieder freizugeben.

Listing 8: Lösung & Bewertung

```
29     free(arr);  // 1 Punkt
```

Aufgabe 4 (9 Punkte) Gültigkeitsbereiche.

Betrachten Sie den folgenden C++ Quellcode:

```
1  static const int z = 23;
2
3  class A {
4      int b = 0;
5
6  public:
7      void foo(int& c) {
8          c = 2 * c - 1;
9      }
10
11     void bar(int d) {
12         for(int i = 0; i < d; i++) {
13             b += i;
14             // Welche Variablen sind hier gültig?
15         }
16     }
17 }
18
19 int main() {
20     int e = 5;
21     A* a = new A();
22     a->foo(e);
23     a->bar(e);
24 }
```

1. (2 Punkte) Auf welche Variablen kann in der markierten Zeile 14 zugegriffen werden?

   Musterlösung: `b, d, i, z`
   0,5 Punkte pro richtiger Antwort, -0,5 Punkte für jede falsche Antwort, jedoch mindestestens 0 Punkte.

2. (1 Punkt) Welche Zeilen in der `main` Funktion beeinflußen den Inhalt von Variable `e`?

   Musterlösung: Zeilen 20 und 22.
   Bewertung: 0,5 Punkte je richtiger Zeile. -0,5 Punkte je falscher Zeile, jedoch mindestens 0 Punkte. Zeilenangaben außerhalb der `main` Funktion werden ignoriert.

3. (1 Punkt) Welche Zeilen in der `main` Funktion beeinflußen den Zustand von dem Objekt auf das `a` zeigt?

   Musterlösung: Zeilen 21 und 23.
   Bewertung: 0,5 Punkte je richtiger Zeile. -0,5 Punkte je falscher Zeile, jedoch mindestens 0 Punkte. Zeilenangaben außerhalb der `main` Funktion werden ignoriert.

4. (5 Punkte) Bestimmen Sie die Gültigkeitsbereiche der Bezeichner in der untenstehenden Tabelle. Geben Sie dazu die erste und letzte Zeilennummer der Gültigkeitsbereiche an. Eine Variable ist in jeder Zeile gültig, in welcher mit einem zusätzlichen Ausdruck (vor oder nach dem gegebenen Ausdruck in dieser Zeile) auf deren Inhalt zugegriffen werden könnte.

   Musterlösung:

Name	gültig von	gültig bis
a	21	24
b	4	17
c	7	9
d	11	16
e	20	24

   0,5 Punkt pro richtigem Eintrag, *keinen* Abzug für falsche Einträge.

Aufgabe 5 (9 Punkte) Vererbung in C++.

Gegeben sei folgende Klassendefinition:

```
1 class Bauelement {
2 private:
3     double spannung;
4 public:
5     Bauelement(double);
6     double getSpannung() {return spannung;}
7     virtual double berechneStrom() = 0;
8 };
```

Der Konstruktor von `Bauelement` sei an anderer Stelle implementiert und initialisiert das Attribut `spannung` mit dem übergebenen **`double`**.

1. (5 Punkte) Deklarieren Sie eine Klasse `Widerstand`, die von `Bauelement` erbt. Diese soll ein privates **`double`** Attribut `R` und einen parametrisierten Konstruktor, der alle Attribute von `Widerstand` mit übergebenen Werten initialisiert, besitzen. Stellen Sie sicher das Objekte von der Klasse `Widerstand` erzeugt werden können!

   Hinweis: Konstruktor und Methoden sollen ausschließlich deklariert werden und werden erst später definiert.

Listing 9: Lösung & Bewertung

```
1 class Widerstand : public Bauelement {  // 1,5 Punkte: Klassendefinition (0,5),
2 private:                                 // Vererbung (0,5), Sichtbarkeit (0,5)
3     double R;                            // 1 Punkt: Typ (0,5), Sichtbarkeit (0,5)
4 public:
5     Widerstand(double, double);          // 1 Punkt: Konstruktor (0,5), Sichtbarkeit (0,5)
6     double berechneStrom();              // 1 Punkt wenn NICHT virtuell
7 };                                       // 0,5 Punkte für { ... };
```

2. (2 Punkte) Implementieren Sie nun den parametrisierten Konstruktor von `Widerstand`. Gehen Sie davon aus, dass Sie sich außerhalb der Klassendefinition (z.B. in einer separaten .cpp Datei) befinden.

Listing 10: Lösung & Bewertung

```
1 Widerstand::Widerstand(double spannung, double r) : // 0,5 Punkte für Namespace
2     Bauelement(spannung),                           // 1 Punkt
3     R(r)                                            // 0,5 Punkte
4 {}
```

3. (2 Punkte) Schreiben Sie die Methode `berechneStrom()` der Klasse `Widerstand`. Gehen Sie wieder davon aus, dass Sie sich außerhalb der Klassendefinition befinden und nutzen Sie folgende Formel:

$$I = \frac{U}{R}$$

   mit I: Strom, U: Spannung und R: Widerstand.

Listing 11: Lösung & Bewertung

```
1 double Widerstand::berechneStrom()       // 0,5 Punkte für Namespace
2 {                                        // 0,5 Punkte für getSpannung()
3     return getSpannung()/R;              // 0,5 Punkte für R
4 }                                        // 0,5 Punkte für korrekte Rückgabe
```

Aufgabe 6 (10 Punkte) Konstrutoren und Destruktoren in C++.

Gegeben ist eine Klasse `Punkt`, welche zweidimensionale Punkte in einem Vektor-Grafik Programm repräsentiert:

```
1 class Punkt {
2 public:
3     float x,y;
4     Punkt() : x(0), y(0) {};
5     Punkt(float x, float y) : x(x), y(y) {};
6 };
```

Ein *Polygon* ist ein Vieleck, welches aus einer beliebigen Anzahl von verbundenen Punkten besteht. Die folgende Klasse definiert ein solches Polygon, ohne bei dessen Entstehung festzulegen, wie viele Punkte es haben wird. Statt dessen kann man enthaltene Punkte mit der Methode `addPunkt` nach und nach hinzufügen.

```
1 class Polygon {
2 private:
3     int anzahl;      // Anzahl von Punkten aus denen das Polygon besteht
4     Punkt* punkte;  // Array mit den Punkten des Polygons
5 public:
6     Polygon();
7     ~Polygon();
8     void addPunkt(float, float);
9 };
```

Hinweis: Es dürfen **keine** Library Funktionen außer **`new`** und **`delete`** benutzt werden, insbesondere kein `vector`, kein `malloc`, kein `calloc` und kein `free`.

1. (2 Punkte) Definieren Sie den parameterlosen Konstruktor von `Polygon`, welcher ein Polygon initialisiert, das noch über keine Punkte verfügt.

   Hinweis: Bei der Entstehung eines Polygons soll die Anzahl der Punkte, welche es letztendlich aufweisen wird, noch nicht feststehen. Sie müssen die Länge des Punkte-Arrays also dynamisch verändern können.

   Listing 12: Lösung & Bewertung

```
1 Polygon::Polygon() : anzahl(0) {    // 1 Punkt für Initalisierung von anzahl
2     punkte = new Punkt[0];          // 1 Punkt für Initalisierung von punkte
3 }              // Achtung: punkte kann auch in addPunkt initalisiert werden!
```

2. (2 Punkte) Definieren Sie den Destruktor von `Polygon`, welcher sicherstellt dass keine Speicherlecks auftreten.

   Listing 13: Lösung & Bewertung

```
1 Polygon::~Polygon() {
2     delete[] punkte;                // 2 Punkte
3 }
```

3. (6 Punkte) Definieren Sie die Methode **`void`** `Polygon::addPunkt(`**`float`** `x, `**`float`** `y)`, welche dem Polygon einen neuen Punkt mit den Koordinaten (`x`,`y`) hinzufügt, d.h. den Array `punkte` um diesen erweitert. Achten Sie darauf, dass die Variable `anzahl` der neuen Anzahl von Punkten entspricht und keine Speicherlecks auftreten.

   Listing 14: Lösung & Bewertung

```
1 void Polygon::addPunkt(float x, float y) {
2     Punkt* arr = new Punkt[anzahl+1];       // 1 Punkt: Neues Array (0,5), Größe (0,5)
3     for (int i = 0; i < anzahl; ++i) arr[i] = punkte[i];    // 2 Punkte: Kopieren (0,5),
4                         // Schleifenvariable init (0,5) und update (0,5), Bedingung (0,5)
5     arr[anzahl] = Punkt(x,y);               // 1 Punkt: auch ohne Konstruktor möglich
6     anzahl++;                               // 0,5 Punkt
7     delete[] punkte;                        // 1 Punkt
8     punkte = arr;                           // 0,5 Punkt
9 }
```

Aufgabe 7 (10 Punkte) Allgemeine Fragen (Rechneraufbau).

In dieser Aufgabe ist jeweils genau eine Antwort richtig, welche Sie ankreuzen sollten. Kreuzen Sie pro Teilaufgabe nur ein Kästchen an. Eine richtige Antwort ergibt einen Punkt, eine falsche 0 Punkte. Es gibt keine Minuspunkte. Um ein versehentlich gesetztes Kreuz wieder zu löschen, füllen Sie das jeweilige Kästchen aus und zeichnen ein leeres daneben.

1. (1 Punkt) Welche Aussage über den Prozessorbus trifft zu:

   ☐ Der Adressbus schreibt und liest Adressen in den Arbeitsspeicher.
   ☒ Der Steuerbus enthält Interrruptsleitungen
   ☐ Ein Bussystem mit höherer Taktrate hat grundsätzlich mehr Bandbreite.
   ☐ Der Grafikbus steutert den Bildschirm an.

2. (1 Punkt) Welche Aussage zu Bereichsüberschreitungen in Ganzzahldarstellungen ist korrekt?

   ☐ Bei Subtraktionen benötigt man mehr als ein Checkbit um sicherzustellen, dass kein Überlauf stattfindet.
   ☐ Bereichsüberschreitungen treten nur bei der Addition bzw. Subtraktion mit Checkbits auf.
   ☒ Die Addition zweiter positiver Zahlen kann ein negatives Ergebnis liefern.
   ☐ Die Zweierkomplementsdarstellung verhindert Bereichüberschreitungen.

3. (1 Punkt) Welche Aussage zu Voll- bzw. Halbaddierern ist **falsch**?

   ☒ Der mögliche Wertebereich eines Volladdierers ist doppelt so groß wie der eines Halbaddierers.
   ☐ Ein Volladdierer besteht aus 2 Halbaddierern und einem OR-Gatter.
   ☐ Je ein XOR- und AND-Gatter bilden einen Halbaddierer.
   ☐ Beide besitzen die gleiche Anzahl von Ausgängen.

4. (1 Punkt) Welche aufsteigende Reihenfolge von Speichern ist korrekt, wenn mit dem kleinsten/schnellsten Speicher begonnen wird?

   ☐ 1. Register, 2. RAM, 3. Cache, 4. Festplatten
   ☐ 1. Festplatten, 2. RAM, 3. Register, 4. Cache
   ☒ 1. Register, 2. Cache, 3. RAM, 4. Festplatten
   ☐ 1. Cache, 2. Register, 3. RAM, 4. Festplatten

5. (1 Punkt) Was ist **keine** Funktion des Betriebssystem-Kerns:

   ☒ Bereitstellen der Eingabeaufforderung.
   ☐ Kontrolle der Hardware.
   ☐ Bereitstellen von Systemfunktionen.
   ☐ Datei & Prozessverwaltung.

6. (1 Punkt) Der Wertebereich von 4-Bit Dualzahlen mit einem Exzess von 2 ist:

   ☒ -2 ... 13
   ☐ 0 ... 15
   ☐ 2 ... 17
   ☐ -7 ... 8

7. (1 Punkt) Welche Aussage zur ALU stimmt **nicht**?

   ☐ Sie dient zur Durchführung von Rechenoperationen (z.B. Addition/Subtraktion).
   ☐ Sie kann booleschen Operationen (z.B. XOR) auswerten.
   ☐ Ist Teil eines Schaltwerk.
   ☒ Sie ist im Steuerwerk untergebracht

8. (1 Punkt) Ein beendeter noch existierender Prozess wird auch bezeichnet als:

   ☐ Leiche
   ☐ Golem
   ☒ Zombie
   ☐ Vampir

9. (1 Punkt) Welches Gesetz bzw. welche Regel gilt **nicht** für die boolesche Algebra?

   ☐ Komplement
   ☐ Idempotenz
   ☐ Absorption
   ☒ Induktion

10. (1 Punkt) Welches Bauelement eignet sich **nicht** zum Speichern?

    ☒ Multiplexer
    ☐ Flip-Flop
    ☐ Binary Cell
    ☐ Register

Aufgabe 8 (6 Punkte) Zahlendarstellung (Rechneraufbau).

1. (2 Punkte) Wandeln Sie die Zahl $48_{(15)}$ (zur Basis 15) in eine Zahl zur Basis 9 um. Der Lösungsweg muss erkennbar sein.

Lösung:

$48_{(15)} = 4 \cdot 15^1 + 8 \cdot 15^0 = 60 + 8 = 68_{(10)}$

$68 : 9 = 7$ *Rest* 5 $(z_0 = 5)$
$7 : 9 = 0$ *Rest* 7 $(z_1 = 7)$

Also ist $48_{15} = 75_9$.

2. (2 Punkte) Wandeln Sie die Dezimalzahl $45_{(10)}$ in eine Hexadezimalzahl um. Der Lösungsweg muss erkennbar sein.

Lösung:

$45 : 16 = 2$ *Rest* 13 $(z_0 = D)$
$2 : 16 = 0$ *Rest* 2 $(z_1 = 2)$

Also ist $45_{10} \ = \ 2D_{16}$.

3. (2 Punkte) Wandeln Sie die Dualzahl $101110,011_{(2)}$ in eine Oktalzahl um. Der Lösungsweg muss erkennbar sein.

Lösung:

$101110,011_{(2)} \ = \ 101|110,011 \ = \ 56,3_{(8)}$

Aufgabe 9 (7 Punkte) Ganzzahldarstellung (Rechneraufbau).

1. (2 Punkte) Wandeln Sie die Dezimalzahl **-13** in eine Zweierkomplementdarstellung mit 6-Bit um. Alle Schritte des Lösungswegs müßen detailliert dargestellt werden.

Lösung:

$-13_{(10)} \rightarrow 110011$
$13_{(10)}$ ins Dualsystem 001101
001101 invertieren $\rightarrow$ 110010
1 addieren $\rightarrow$ 110011

2. (1 Punkt) Wandeln Sie die Zahl in 6-Bit-Zweierkomplementdarstellung (**011011**) in eine Dezimalzahl um. Alle Schritte des Lösungswegs müßen detailliert dargestellt werden.

Lösung:

$011011 \rightarrow 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^1 + 1 \times 2^0 \rightarrow 16 + 8 + 2 + 1 = 27$

3. (4 Punkte) Führen Sie die nachfolgenden Operation mit Hilfe des 6-Bit Zweierkomplements aus. Testen Sie auf Überlauf/Unterlauf und geben Sie an um welchen es sich handelt, sollte es zu einem kommen. Wandeln Sie das Ergebnis in jedem Fall wieder in Dezimaldarstellung um. Der Lösungsweg muss detailliert dargestellt werden:

$$-13_{(10)} + 27_{(10)}$$

Lösung:

Binär:	110011
	011011
+ Checkbit	[1]110011
	[0]011011
Übertr.:	1[1]100110
Summe	1[0]001110
Check:	ok

Kein Unter/Überlauf, da das Checkbit mit dem Vorzeichenbit des Ergebnisses übereinstimmt: `[0]0`. Das Ergebnis 14 passt also in eine 6-Bit Zweierkomplement Zahl.

$001110 \rightarrow 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 = 8 + 4 + 2 = 14$

$011011 \rightarrow 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^1 + 1 \times 2^0 \rightarrow 16 + 8 + 2 + 1 = 27$

Aufgabe 10 (10 Punkte) Gleitkommadarstellung (Rechneraufbau).

1. (4 Punkte) Wandeln Sie folgende rationale Zahl $-5.45_{(10)}$ in die Gleitkommadarstellung um. Der Lösungsweg muss dabei detailliert erkennbar sein.

   Hinweis: Es soll folgendes Darstellungsformat angenommen werden: (1Bit Vorzeichen, 4 Bit Exponent, 5 Bit Mantisse, Exzess = 7)

   Lösung:

   $-5 \rightarrow -101$
   $0,45 \rightarrow 0.01110011$
   $0,45 * 2 = 0,9 (z_{-1} = 0)$
   $0,9 * 2 = 1,8 (z_{-2} = 1)$
   $0,8 * 2 = 1,6 (z_{-3} = 1)$
   $0,6 * 2 = 1,2 (z_{-4} = 1)$
   $0,2 * 2 = 0,4 (z_{-5} = 0)$
   $0,4 * 2 = 0,8 (z_{-6} = 0)$
   $-5.45 \rightarrow -101.011100$
   $-1.01011100 * 2^2$
   Vorzeichen = 1
   Exponent $= 2 + 7 = 9 \rightarrow 1001$
   Verk. Mantisse = .01011
   $\Rightarrow$ 1100101011

2. (6 Punkte) Addieren Sie die Gleitkommazahlen 0 0101 10110 und 0 0011 01001. Wie groß ist der dort entstandene Rundungsfehler? Der Rundungsfehler entspricht der Wertigkeit der wegfallenden Stellen und kann in Form einer Zweierpotenz angegeben werden.

   Hinweis: Es soll folgendes Darstellungsformat angenommen werden: (1Bit Vorzeichen, 4 Bit Exponent, 5 Bit Mantisse, Exzess = 7)

   Lösung:

   $1.10110 \cdot 2^{5-7} + 1.01001 \cdot 2^{3-7}$
   Exponent anpassen: $1.10110 \cdot 2^{5-7} + 0.0101001 \cdot 2^{5-7}$
   Genauigkeitsverlust: $0.0000001 \cdot 2^{-2} \rightarrow 2^{-9}$
   Addieren:

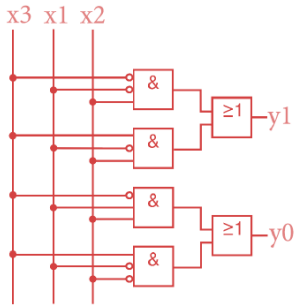
1.10110	00
+0.01010	01
10.00000	01

   $\Rightarrow 1.0 \cdot 2^{6-7}$
   Vorzeichen: 0
   Exponent: 0110
   Verk. Mantisse : 00000
   Ergebnis $\Rightarrow$ 0011000000

Aufgabe 11 (7 Punkte) Schaltungen & Boolesche Ausdrücke (Rechneraufbau).

1. (3 Punkte) Beschriften Sie die Ein- und Ausgänge der nachfolgenden Schaltung, so dass die Tabelle erfüllt ist.

Lösung:



x_0	x_1	x_2	y_0	y_1
0	0	0	0	0
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	0	0
1	0	1	0	0
1	1	0	1	0
1	1	1	0	0

2. (4 Punkte) Überprüfen Sie mittels einer Wertetabelle ob der gegebene boolesche Ausdruck zu y_0 oder y_1 äquivalent ist.

$$(x_2 \oplus x_0)\,(\overline{x}_0\overline{x}_1 + x_0x_1)$$

Lösung:

x1	x2	x3	$(x3 \oplus x1) * ((\overline{x1} * \overline{x2}) + (x1 * x2))$				
0	0	0	0	0	1	1	0
0	0	1	1	1	1	1	0
0	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	1	0	0	0	0
1	0	1	0	0	0	0	0
1	1	0	1	1	0	1	1
1	1	1	0	0	0	1	1

↑

=> Ist Äquivalent zu y0 !