Matrikelnummer: Name: #:

Hörsaal: Sitzplatznummer:

Prof. Dr.-Ing. K. D. Müller-Glaser

Informationstechnik

WS 2010/11

Institut für Technik der Informationsverarbeitung, KIT

Klausur WS 2010/11

Mo., 14.02.2011

Hinweise zur Klausur

Hilfsmittel

Zur Prüfung ist nur eine handgeschriebene 2-seitige A4 Formelsammlung zugelassen. Nicht erlaubt sind die Verwendung eines Mobiltelefons und jegliche Kommunikation mit anderen Personen.

Prüfungsdauer

Die Prüfungsdauer beträgt 120 Minuten.

Prüfungsunterlagen

Die Prüfungsunterlagen bestehen aus insgesamt 26 Seiten Aufgabenblättern (diesem Titelblatt, 8 Aufgabenblöcken, 2 zusätzlichen Lösungsblättern)

Bitte kontrollieren Sie vor der Bearbeitung der Aufgaben auf jeder Seite oben Ihren Namen und Ihre Matrikelnummer!

Falls Sie zusätzliche Blätter zur Lösung der Aufgaben benötigen, verwenden Sie die zusätzlichen Lösungsblätter bzw. fragen Sie nach zusätzlichem Papier. Auf jedes zusätzliche Lösungsblatt ist neben dem Namen auch die Aufgabennummer mit einzutragen. Vermeiden Sie das Beschreiben der Rückseiten. Die Verwendung von eigenen Blättern ist nicht erlaubt. Am Ende der Prüfung sind die 26 Seiten Aufgaben- und Lösungsblätter und alle zusätzlichen Lösungsblätter im ausgehändigten Umschlag abzugeben. Verwenden Sie zum Bearbeiten der Aufgaben nur dokumentenechte Schreibgeräte – keinen Bleistift, keine Rotstifte!

Wie verabredet enthält die Klausur mehr Aufgaben als, bei einer richtigen Lösung, zum Erreichen einer sehr guten Note (1,0) notwendig sind (Nutzen Sie die Auswahlmöglichkeit). Bitte beachten Sie: Die mit * gekennzeichneten Teilaufgaben können Sie nicht unabhängig von den anderen Teilaufgaben lösen.

Aufgabe	1	2	3	4	5	6	7	8	Σ
≈ Gewichtung ca. [%]	13	10	10	13	13	12	13	16	100
Ergebnis [P]									

Aufgabe 1 Allgemeine Fragen

Beantworten Sie folgende Fragen:

A) Nach welcher Größe richtet sich die Breite des Adressbusses eines Speichers bei der parallelen Ansteuerung?

B) Was versteht man unter der Abkürzung ALU im Gebiet der Rechnerarchitektur?

C) Nennen Sie zwei Vorteile für den Programmierer durch die Verwendung von höheren Programmiersprachen gegenüber der Verwendung des Maschinenbefehlssatzes?

D) Nennen Sie drei Beispiele für objektorientierte Programmiersprachen.

E) Berechenbarkeitstheorie besagt, dass: (bitte ankreuzen)

Bitte beachten Sie, dass falsche Antworten einen Punktabzug bedeuten. Negative Punkte werden allerdings nicht auf andere Teilaufgaben übertragen.

Antwort:	**Wahr**	**Falsch**
der Algorithmus bei denselben Voraussetzungen das gleiche Ergebnis liefern muss.		
jeder Schritt des Verfahrens tatsächlich ausführbar sein muss.		
ein Algorithmus immer terminiert. (der Algorithmus überhaupt jemals erfolgreich beendet werden kann)		

F) Nennen Sie die vier Schritte in einem Software-Entwicklungsprozess in der richtigen Reihenfolge.

G) Beim Simulated-Annealing Algorithmus kann es bei mehrmaligem Ausführen zu unterschiedlichen Endergebnissen kommen. Warum?

H) In der folgenden Tabelle sind typische Eigenschaften von objektorientierten Programmiersprachen aufgelistet. Geben Sie eine kurze Erklärung der Begriffe in der Tabelle.

Begriff	**Erklärung**
Abstraktion	
Kapselung	
Polymorphie	
Vererbung	

I) In der abschließenden Software Testphase werden verschiedene Tests durchgeführt. Beschreiben Sie kurz die angegebenen Tests und benennen Sie die Ziele?

Tests	**Beschreibung / Ziel**
Alpha Test	**Beschreibung:** **Ziel:**
Beta Test	**Beschreibung:** **Ziel:**

Aufgabe 2 C++ Verständnisfragen

A) Vervollständigen Sie in der folgenden Tabelle das Bitmuster des linken Operanden jeder Operation (**nach** der Ausführung dieser Operation). Die dargestellten C++ Operationen werden nacheinander ausgeführt.

`unsigned char a, b, c;`	Bitmuster							
`a = 7;`	0	0	0	0	0	1	1	1
`b = 85;`	0	1	0	1	0	1	0	1
`c = ~b;`								
`c = a << 3;`								
`c = a | b;`								
`c = b & 254;`								
`c = a ^ b;`								
`c = b * 2;`								

B) Gegeben ist der folgende C++ Programmausschnitt. Nehmen Sie an, dass das Programm korrekt und fehlerfrei kompiliert worden ist. Welche Ausgaben erfolgen auf dem Bildschirm?

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

int main() {
  int zahl1 = 10;
  int zahl2 = 10 / 4;
  float zahl3 = 4.0;
  cout << zahl1++ << endl;          _____________
  cout << zahl2 << endl;            _____________
  cout << --zahl1 / zahl3 << endl;  _____________
  cout << zahl1 % 10 << endl;       _____________
  return 0;
}
```

C) Gegeben ist der folgende C++ Programmausschnitt. Nehmen Sie an, dass das Programm korrekt und fehlerfrei kompiliert worden ist. Welche Ausgabe erfolgt auf dem Bildschirm?

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

int main() {
  int var = 0;
  for( int i = 0; i <= 99; i++ ) {
    for( int j = 1; j < 100; j++ ) {
      var++;
    }
    var++;
  }

  cout << "var: " << var << endl;
  return 0;
}
```

Ausgabe:

D) Gegeben ist das folgende C++ Programm. Dieses enthält Programmzeilen, welche sowohl beim kompilieren, als auch zur Laufzeit zu Fehlern führen können. Benennen Sie drei mögliche Fehlerquellen im C++ Programmcode.

```
#include <iostream>

int main() {
  int v = 3;
  array1[5] = {1, 2, 3, 4, 5};
  v = array1[5];
  cout << "Hallo" << Welt;
  return 0;
}
```

1) _____

2) _____

3) _____

E) Warum kann der folgende C++ Code zu Laufzeitfehlern führen?

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

int* calc() {
  int j = 7;
  return &j;
}

int main() {
  int i = 9;
  int* ip = &i;
  *ip = 8;
  ip = calc();
  // weiterer Code
  *ip = 8;
  cout << *ip;
  return 0;
}
```

Antwort:

G) Richtig oder falsch?

Bitte beachten Sie, dass falsche Antworten einen Punktabzug bedeuten. Negative Punkte werden allerdings nicht auf andere Teilaufgaben übertragen.

i) Eine Referenz repräsentiert die Adresse und den Typ eines Objekts.

☐ Falsch ☐ Richtig

ii) Der Operator `&&` verknüpft zwei Operanden und gibt `true` zurück, wenn beide Operanden den Wert `true` haben, sonst `false`.

☐ Falsch ☐ Richtig

iii) Der Vorteil einer verketteten Liste gegenüber einem dynamisch angelegten Array ist, dass die verkettete Liste weniger Speicherplatz belegt.

☐ Falsch ☐ Richtig

iv) Der Operator `new` erwartet als Operand den Typ des anzulegenden Objekts.

☐ Falsch ☐ Richtig

Aufgabe 3 Objektorientierung in C++

A) Erklären Sie kurz den Unterschied zwischen…

i) der Deklaration und der Definition einer Variablen?

ii) einer lokalen und einer globalen Variablen?

iii) einer ‚public' und einer ‚private' Klassenmethode?

B) Vervollständigen Sie die Definitionen der Methoden `toString()` im folgenden C++ Programm, sodass sich das Programm fehlerfrei kompilieren und ausführen lässt. Beachten Sie dabei folgende Randbedingungen:

- Verwenden Sie die vorgegebenen Variablen und Methoden, sowie deren Namen. Sie dürfen dabei keine neuen Methoden und Variablen definieren.
- Die Methoden `toString()` der drei Klassen sollen **alle** in dem entsprechenden Objekt verfügbaren Daten (Inhalte der Attribute) in einen String umwandeln und zurückgeben. → Ergänzen Sie die Methoden auf der übernächsten Seite innerhalb der entsprechenden Vorlagen.
- Die folgende Abbildung zeigt die Ausgabe des Programms, welche erzeugt werden soll.

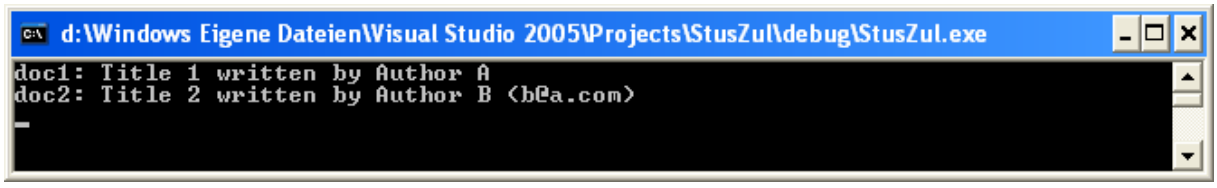


Abb. 3-1 Ausgabe des Programms

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

class Author {
  public:
    Author() {}

    Author( string authorName ) {
      name = authorName;
    }

    string toString();

    virtual bool IhaveEmail() {
      return false;
    }

  protected:
    string name;
};

class AuthorWithEmail : public Author {
  public:
    AuthorWithEmail( string authorName, string authorEmail ) {
      email = authorEmail;
      Author::name = authorName;
    }

    string toString();

    bool IhaveEmail() {
      return true;
    }

  private:
    string email;
};

class Document {
  public:
    Document( Author* docAuthor, string docTitle ) {
      author = docAuthor;
      title = docTitle;
    }

    string toString();

  private:
    Author* author;
    string title;
};

int main() {
  Author* authorA = new Author( "Author A" );
  AuthorWithEmail* authorB = new AuthorWithEmail( "Author B", "b@a.com" );
  Document* doc1 = new Document( authorA, "Title 1" );
  Document* doc2 = new Document( authorB, "Title 2" );

  cout << "doc1: " << doc1->toString() << endl;
  cout << "doc2: " << doc2->toString() << endl;
  return 0;
}
```

Ergänzen Sie die Methoden auf der nächsten Seite!

Ergänzen Sie die folgenden Methoden:

```
string Author::toString() {




}

string AuthorWithEmail::toString() {




}

string Document::toString() {
  string docInfos = title + " written by ";









}
```

C) Kann im obigen C++ Programmcode das Attribut `name` der Klasse `Author` als `private` deklariert werden? Begründen Sie Ihre Antwort.

Aufgabe 4 Datenstrukturen

Für jede objektorientierte Programmiersprache wird eine Möglichkeit benötigt, dynamisch Speicher anzufordern und wieder freizugeben. Im Folgenden soll daher eine einfache Implementierung einer dynamischen Speicherverwaltung erstellt werden.
Die Klasse `MyMem` dient hierbei zur Verwaltung eines Speicherreservoirs, aus dem Speicherblöcke angefordert werden können. Sie stellt die folgenden öffentlichen Methoden zur Verfügung:

- Die Methode `requestMem` dient zur Anforderung eines Speicherblocks
- Die Methode `freeMem` gibt einen zuvor angeforderten Speicherblock frei

Als Speicherreservoir dient ein Array vom Typ *unsigned char* mit der Länge 256.

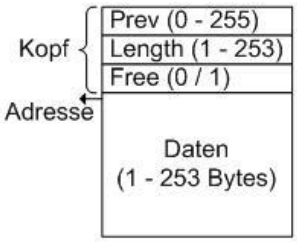


Abb. 4-1 Listenelement

Zur Verwaltung der Speicherblöcke im Speicherreservoir dient eine verkettete Liste. Jeder belegte und freie Speicherblock wird durch ein Element dieser Liste beschrieben.

Wie in Abb. 4-1 dargestellt, besteht ein solches Element aus aufeinander folgender Bytes in dem Array. Jedes Element besteht aus einem 3 Bytes langem Kopf, der Informationen zur Verwaltung der Liste enthält, gefolgt von einem Datenbereich mit einer Länge von 1 bis 253 Bytes. Die drei Bytes des Kopfs sind in der Abbildung mit *Prev*, *Length* und *Free* bezeichnet. *Prev* enthält den Array-Index des Vorgängerelements (falls es keinen Vorgänger gibt wird hier 255 eingetragen), *Length* enthält die Länge des Datenteils (d.h. ohne Kopf) und *Free* gibt an, ob das Element gerade verwendet wird (0) oder Teil des freien Speichers ist (1). Das Array ist immer komplett mit Elementen ausgefüllt (siehe Abb. 4-2).

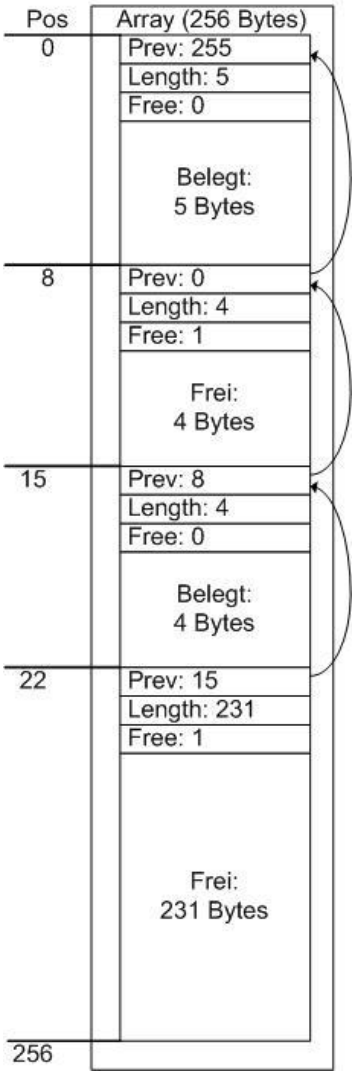


Abb. 4-2 Liste

Die Methode `requestMem` durchsucht die Liste nach einem freien Element mit ausreichender Länge. Beim Durchsuchen ergibt sich der Index des nächsten Elements aus dem Index des aktuellen Elements plus der Länge des aktuellen Elements inkl. Kopf. Sobald ein Element gefunden wird, wird es belegt und ein Zeiger auf den Beginn des Datenblocks zurückgegeben.
Falls der Datenbereich des gewählten Elements mehr als 3 Bytes größer ist als die angeforderte Länge, wird ein neues freies Element dahinter eingefügt. Andernfalls wird die Länge des Elements nicht verändert und das Element komplett als belegt markiert.

Die Methode `freeMem` dient zum Freigeben eines Elements und erhält dabei als Übergabeparameter einen Zeiger auf den Beginn des Elements. Beim Freigeben wird überprüft, ob unmittelbare vorhergehende oder nachfolgende Elemente ebenfalls frei sind. Ist dies der Fall, müssen die angrenzenden freien Elemente zu einem einzigen freien Element zusammengefasst werden.

Eine komplette Übersicht der Attribute und Methoden der Klasse `MyMem` können Sie der nachstehenden Definition entnehmen.

```
class MyMem {
  public:
    MyMem ();
    ~MyMem ();

    unsigned char* requestMem( unsigned char size );
    void freeMem( unsigned char* address );
  private:
    unsigned char buffer[256];

    unsigned char getNextElement( unsigned char pos );
    unsigned char getPrevElement( unsigned char pos );
    unsigned char findFreeElement( unsigned char size );

    void writeHeader( unsigned char pos,
                      unsigned char prev,
                      unsigned char length,
                      bool free );
};
```

Beschreibung der bereits implementierten Methoden:

- `getNextElement`:
  Gibt die Startposition des nachfolgenden Elements zurück. Falls das aktuelle Element das Letzte in der Liste ist, wird `255` zurückgegeben. `pos` enthält die Startposition des aktuellen Elements.

- `getPrevElement`:
  Gibt die Startposition des vorhergehenden Elements zurück. Falls das aktuelle Element das Erste in der Liste ist, wird `255` zurückgegeben. `pos` enthält die Startposition des aktuellen Elements.

- `writeHeader`:
  Schreibt den Inhalt der übergebenen Variablen `prev`, `length` und `free` in die entsprechenden Felder des Kopfs des Elements an Position `pos`.

A) Nehmen Sie an, dass die Methode `requestMem` bei einem Aufruf ein freies Element mit ausreichender Länge gefunden hat. Warum ist dabei in der Methode eine Unterscheidung notwendig, ob innerhalb des gefundenen Elements ein zusätzliches freies Element eingefügt wird oder nicht?

B) Warum ist es wichtig, dass in der Methode `freeMem` angrenzende freie Elemente zusammengefasst werden?

C) Zum Durchsuchen der Liste nach einem freien Element dient die Methode `findFreeElement`. Dazu wird die Liste so lange durchsucht, bis ein freies Element gefunden wurde, welches mindestens die angeforderte Länge (Übergabeparameter `size`) aufnehmen kann. Wird ein solches Element gefunden, wird dessen Anfangsposition zurückgegeben, falls nicht wird 255 zurückgegeben. Zeichnen Sie zur Methode ein Nassi-Shneiderman-Diagramm. Sie können dabei bereits vorhandene Methoden verwenden.

D) Nachfolgend finden Sie den C++ Quellcode der Methode `requestMem`. Setzen Sie die Buchstaben der mit A-G gekennzeichneten Codeabschnitte in die passenden Kästen ein, damit die Methode die beschriebene Funktionalität erfüllt.

A	`unsigned char pos = findFreeElement( size );`
B	`return 0;`
C	`size += freeLength;` `freePos = pos;`

D	`unsigned char length = buffer[pos+1];`
	`unsigned char freePos = pos + size + 3;`
	`unsigned char nextPos = getNextElement( pos );`
	`unsigned char freeLength;`

E	`freeLength -= 3;` `writeHeader( freePos, pos, freeLength, true );`
F	`return &buffer[pos+3];`

G	`writeHeader( pos, getPrevElement( pos ), size, false );`
	`if( nextPos != 255 ) {`
	`  writeHeader( nextPos, freePos, buffer[nextPos+1],`
	`  buffer[nextPos+2] );`
	`}`

```
unsigned char* MyMem::requestMem( unsigned char size ) {
  if( size == 0 ) {
    return 0;
  }
  [ ]
  if( pos != 255 ) {
    [ ]
    if( nextPos == 255 ) {
      freeLength = 256 - freePos;
    } else {
      freeLength = nextPos - freePos;
    }

    if( freeLength < 4 ) {
      [ ]
    } else {
      [ ]
    }
    [ ]
    [ ]
  }
  [ ]
}
```

Aufgabe 5 Dateiverarbeitung in C++

In Abbildung 5.1 ist die Literaturliste der Vorlesung IT als Datei „`order.txt`“ gegeben. Dabei werden die einzelnen Datenfelder durch Semikolon getrennt. Zeilenweise sind jeweils Autor, Titel, Verlag und Preis eines Buches angegeben.
Die folgenden Teilaufgaben beziehen sich auf diese Datei. Des Weiteren gilt für alle Teilaufgaben, dass verwendete Variablen vorher deklariert werden müssen. Implementieren Sie alle Teilaufgaben in der Programmiersprache C++.

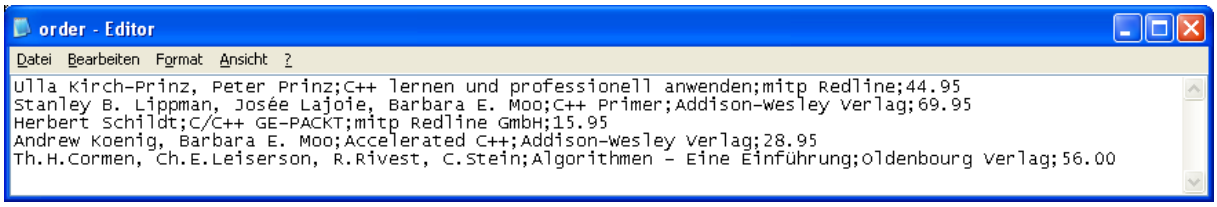


Abb. 5-1 Beispiel Literaturliste

Die Literaturliste soll in einer doppelt-verketteten Liste gespeichert werden, dabei gilt für diese die folgende allgemeine Struktur:

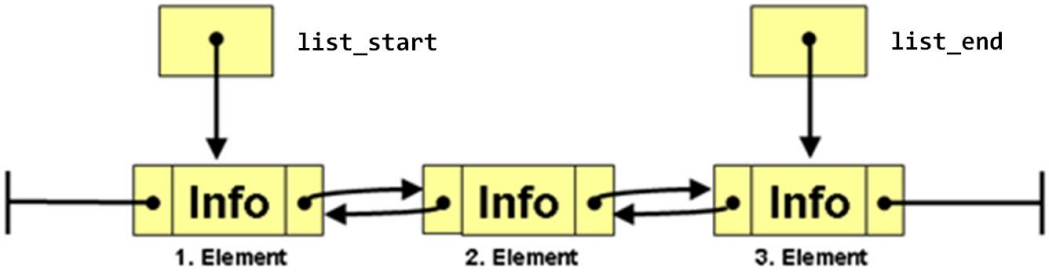


Abb. 5-2 Allgemeine Struktur der doppelt-verketteten Liste

Weiterhin ist die Struktur der Klasse zur Verwaltung der doppelt-verketteten Liste gegeben:

```
class Bibliothek {
  public:
    Book* list_start;
    Book* list_end;

    Bibliothek() {
      list_start = NULL;
      list_end = NULL;
    }
    void readfile();
    void addbook( string zeile );
    void specialFunc();
    void deleteAllElements();
};
```

Hinweis:
Beschreibung der Methode `substr()` der Klasse `string`:

`string st;`
`st.substr( i )`: gibt eine Teilzeichenkette des Strings `st` von Index `i` bis zum Ende zurück

`st.substr( i, n )`: gibt eine Teilzeichenkette des Strings `st` von Index `i` mit max. Länge `n` zurück

A) Schreiben Sie die Programmzeilen, die folgende Aufgaben erfüllen. Die Datei *„info.txt“* soll zum Schreiben geöffnet werden. Diese soll dabei als `output_file` bezeichnet werden. Überprüfen Sie, ob das Öffnen erfolgreich war und geben Sie eine entsprechende Fehlermeldung aus. Schreiben Sie in die neue Datei nach dem erfolgreichen Öffnen den String `"Hallo Welt"`. Schließen Sie die Datei am Ende.

B) Deklarieren Sie die Klasse `Book`, sodass alle angegebenen Datenfelder separat abgespeichert werden können. Der Preis soll dabei als `double` in der Klasse gespeichert werden. Deklarieren Sie zusätzlich die notwendigen Elemente, damit die Elemente der Klasse eine doppelt-verkettete Liste bilden können. Zur Vereinfachung dürfen Sie alle Attribute als `public` deklarieren. Der Startzeiger und der Endezeiger der Liste sind in der bereits beschriebenen Klasse `Bibliothek` enthalten. Die Klasse `Book` soll zusätzlich einen leeren Konstruktor und Destruktor haben.

```
class Book {
  public:




};
```

C) *Implementieren Sie die Methode `addbook` der Klasse `Bibliothek` zum Einfügen eines neuen Elements in die verkettete Liste. Die Methode bekommt dabei immer eine komplette Zeile aus der Datei (Abb. 5-1) als String übergeben. Die Methode soll ein neues Listenelement `Book` anlegen und den einzelnen Attributen die entsprechenden Werte zuweisen. Beachten Sie dabei die allgemeine Struktur der verketteten Liste in Abb. 5-2.

```
void Bibliothek::addbook(string zeile) {




}
```

D) *Schreiben Sie die Methode `specialFunc` der Klasse `Bibliothek` in Pseudocode. Diese hat keine Übergabeparameter und soll alle Bücher vom Verlag „Addison-Wesley“ finden und deren Gesamtpreis ausgeben. Wenn keine Bücher in der Liste oder keine Bücher des Verlags vorhanden sind, soll jeweils ein entsprechender Hinweis auf der Konsole ausgegeben werden.

Aufgabe 6 Graphentheorie und Tiefensuche

Gegeben ist der folgende Graph Z:

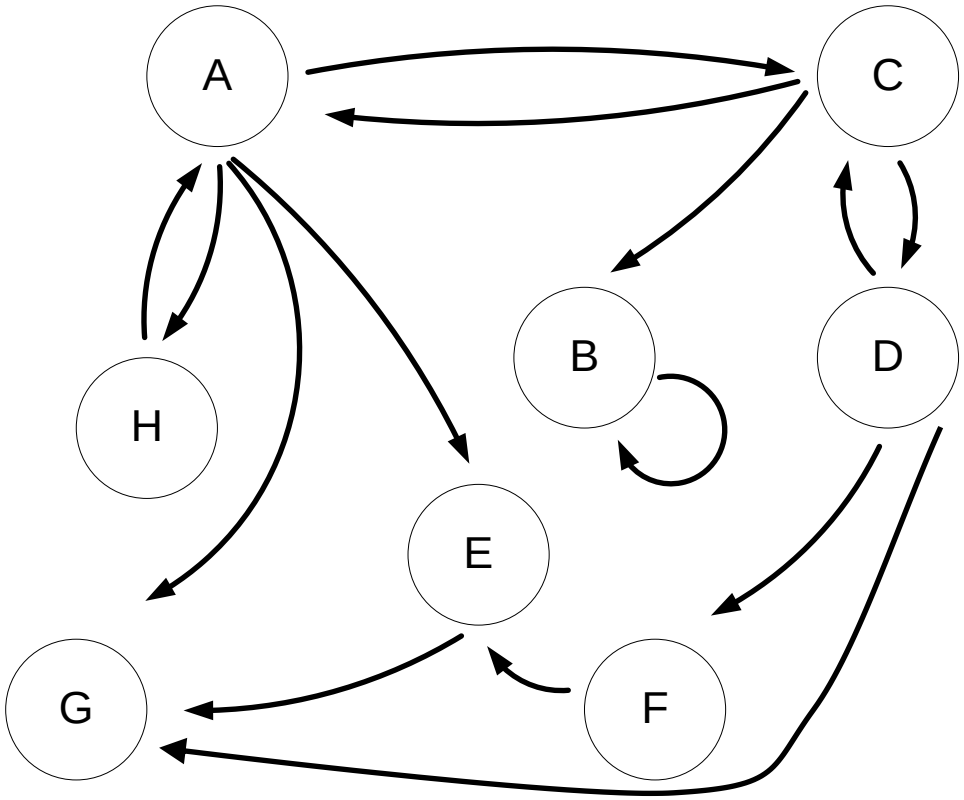


Abb. 6-1 Graph Z

Gegeben ist weiterhin der Pseudocode des Tiefensuchalgorithmus:

```
DFS( Z )
  for alle Knoten u in Z
  do farbe[u] = weiss
     vater[u] = NULL
  zeit = 0
  for alle Knoten u in Z
  do if farbe[u] == weiss
     then DFS-VISIT( u )
```

```
DFS-VISIT( u )
  farbe[u] = grau; zeit = zeit + 1
  startTime[u] = zeit
  for alle Knoten v aus Adj[u]
  do if farbe[v] == weiss
     then vater[v] = u
          DFS-VISIT( v )
  farbe[u] = schwarz; zeit = zeit + 1
  endTime[u] = zeit
```

Dabei gilt:

- `Z:` gegebener Graph
- `Adj[u]:` Alle Nachfolger des Knotens `u`

A) Die folgenden Fragen beziehen sich auf den gegebenen Graphen Z. Bitte kreuzen Sie jeweils die richtige Antwort an und begründen Sie diese.
Bitte beachten Sie, dass Sie Teilpunkte nur bei einer korrekten Begründung erhalten.

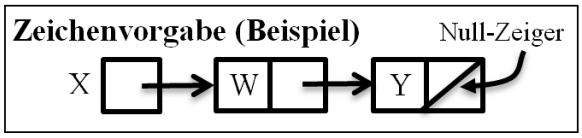
i) Der Graph ist:
☐ ein Baum ☐ kein Baum
Begründung:

ii) Der Graph ist:
☐ gerichtet ☐ ungerichtet
Begründung:

iii) In der Adjazenzmatrix gibt es pro Spalte:
☐ maximal 4 mal '1'-Stellen ☐ maximal 2 mal '1'-Stellen
Begründung:

iv) Der Graph ist:
☐ schleifenfrei ☐ nicht schleifenfrei
Begründung:

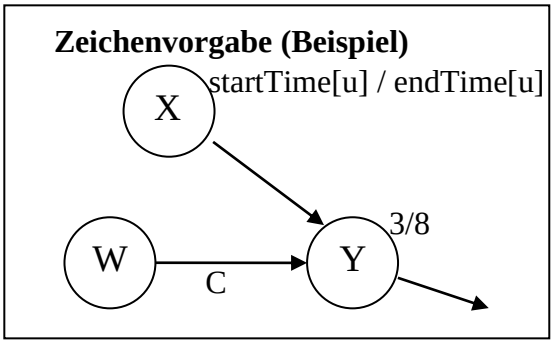
B) Zeichnen Sie die dem Graphen Z entsprechende Adjazenzliste. Die Reihenfolge der Liste soll dabei alphabetisch erfolgen.



C) Wenden Sie den Tiefensuchalgorithmus (DFS) auf den Graphen Z (Seite 18) an, um festzustellen, ob ausgehend vom Knoten A der Knoten G erreichbar ist. Arbeiten Sie den DFS-Algorithmus mit Startknoten A vollständig bis zum Ende ab. Zeichnen Sie den dabei entstehenden Tiefensuchbaum bzw. Tiefensuchwald.

- Die Entdeckungsreihenfolge ist dabei bestimmt durch das Alphabet.
- Bestimmen Sie für jeden Knoten die Entdeckungszeit startTime[u] und die Endzeit endTime[u], endsprechend dem gegebenen Algorithmus. Geben Sie diesen entsprechend der Zeichenvorgabe im Baum an.

Zeichenvorgabe (Beispiel)



- Zeichnen Sie alle fehlenden Kanten zusätzlich in den aus der Tiefensuche entstandenen Baum bzw. Wald ein. Markieren Sie die Kanten wie folgt:

Baumkanten:	keine Markierung
Vorwärtskanten:	mit einem F (Forward)
Rückwärtskanten:	mit einem B (Backward)
Querkanten:	mit einem C (Cross)

Aufgabe 7 Algorithmus-Analyse: Sortieren

Gegeben ist ein Echtzeitsystem, in dem Zahlen in einem Array sortiert werden. Zum Sortieren wird der einfache, deterministische Algorithmus Bubblesort (deutsch „Blasensortierung“) angewendet. Um die Echtzeitbedingung nicht zu verletzten, muss die maximale Größe des Arrays begrenzt werden. Dazu ist eine Feinlaufzeitanalyse notwendig.

Der relative Aufwand für verschiedene Aktionen und Operationen ist in untenstehender Tabelle zusammengefasst.

Elementare Aktion/Operation	**Kosten (Rel. Zeitaufwand)**
Zuweisung	1,0
Addition/Subtraktion	1,4
Multiplikation	2,3
Division	8,0
Vergleich (inkl. Sprung bei *if* oder *while*)	1,5
Indizierung einer Matrix/Array	3,2

A) Führen Sie die Feinlaufzeitanalyse für den **Worst Case Fall** auf den nächsten Seiten entsprechend aus.

Hinweis: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Schreiben Sie in der Spalte „Anzahl von Ausführungen“ den Ausdruck für die Anzahl von Iterationen der entsprechenden Zeile abhängig von der Arraygröße **n**.

Geben Sie in der Spalte Gewicht die Gesamtsumme der Kosten für die Aktion/Operation der entsprechenden Zeile an. Achten Sie darauf, dass in einer Zeile mehrere elementare Aktionen und Operationen möglich sind.

Die Spalten A1, A2 und A3 können Sie als Hilfestellung verwenden, um die einzelnen Ausdrücke in elementarere Ausdrücke aufzuspalten und die Formel auf Seite 23 zu erstellen. Ein Ausfüllen dieser Spalten ist allerdings nicht verlangt (und wird nicht bewertet).

Worst case Betrachtung

Zeile		Anzahl von Ausführungen	Kosten	Zerlegung		
---	`int i, j, n; int A[n];` `// weitere Zeilen`	---	---	A1 $\frac{n^2-n}{2}$	A2 n	A3 konstant
1	`i = 0;`	1				
2	`while ( i < n - 1 ) {`	n				
3	`  j = 1;`	$n-1$				
4	`  while ( j < n - i ) {`	$\frac{n(n+1)}{2}-1$				
5	`  if ( A[j-1] > A[j] ) {`	$\frac{n(n-1)}{2}$				
6	`  temp = A[j-1];`	$\frac{n(n-1)}{2}$				
7	`  A[j-1] = A[j];`	$\frac{n(n-1)}{2}$				
8	`  A[j] = temp;` `  }`	$\frac{n(n-1)}{2}$				
9	`  ++j;` `  }`	$\frac{n(n-1)}{2}$				
10	`  ++i;` `}`	$n-1$				

B) *Schreiben Sie den Ausdruck für die Laufzeit `t` als Funktion der Arraygröße **n**. Dieser muss nicht vereinfacht werden.

`t =`

C) *Betrachten Sie die erste `while` – Schleife:

```
while ( i < n - 1 )
```

Wie können Sie diese Zeile optimieren, um bei größeren Arrays die Laufzeit zu verkürzen? Schreiben Sie den entsprechenden C++ Programmcode. Wie groß wäre der Gewinn an Laufzeit?

Aufgabe 8 Rechnerarchitektur

A) Allgemeine Fragen

i) Erklären Sie den Unterschied zwischen einer Akkumulatormaschine und einer Registermaschine bezüglich der arithmetischen Funktion ADD.

ii) In **Abb. 8–1** ist der Programmcode einer Registermaschine im ISA Befehlsformat „Maschine Register-Memory“ gegeben. Welche Operation wird durch den Code ausgeführt?

```
Load Reg3,B
Load Reg4,D
Add Reg4,C
Sub Reg3,Reg4
Store A,Reg3
```

Abb. 8–1: Programmcode

iii) Wandeln Sie den Code aus **Abb. 8–1** in das ISA Format einer Stackmaschine um.

B) Cache-Strukturen

Ein Prozessor mit einer 16 Bit Architektur adressiert seinen Hauptspeicher mit 28 Bit. Es soll nun ein Cachebaustein in SRAM Technologie integriert werden, der 512 kByte Daten aus dem Hauptspeicher aufnehmen kann. Der Cache ist als 2-Weg assoziativ Cache ausgeführt und kann pro Cacheadresse 64 Bit Daten aus dem Hauptspeicher aufnehmen.

i) Ergänzen Sie die gestrichelten Felder im unten stehenden Diagramm, welches den zu integrierenden Cache darstellt. Bitte beachten Sie, dass Sie nur in Felder, auf welche eine passende Einheit folgt, eine Zahl als Lösung eintragen, ansonsten sind nur Begriffe gefragt.

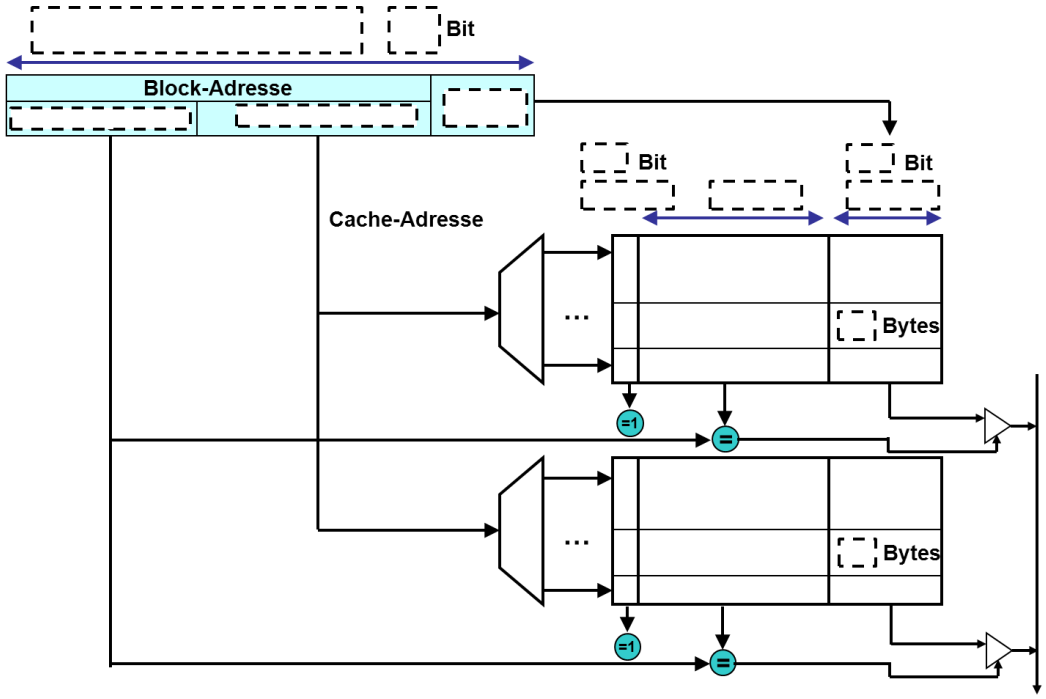


Abb. 8–2: 2-Weg assoziativ Cache

ii) *Wie viele Bits werden für den **Offset** benötigt? Geben Sie einen nachvollziehbaren Rechenweg an.

iii) *Wie viele Bits werden für den **Index** benötigt? Geben Sie einen nachvollziehbaren Rechenweg an. *Hinweis: Stellen Sie, wenn immer möglich, Zahlen als 2er Potenz dar*.

iv) *Wie groß ist der tatsächliche Speicherbedarf des gesamten Caches? Geben Sie einen nachvollziehbaren Rechenweg an. Sie müssen das Ergebnis nicht kürzen.

v) Welche Cache-Art ermöglicht die kürzeste, durchschnittliche Zugriffszeit? Begründen Sie Ihre Antwort.