Informationstechnik
Klausur SS 2013

Institut für Technik der Informationsverarbeitung – ITIV
Prof. Dr.-Ing. Klaus D. Müller-Glaser

Informationstechnik
Datum: 02.10.2013
Name: «Vorname» «Nachname»
Matrikelnummer: «Matrikelnummer»
#: «LaufNr»

Hörsaal: Neue Chemie
Platznummer: «PlatzNr»

Hinweise zur Klausur

Hilfsmittel

- Erlaubte Hilfsmittel sind Lineal und eine handgeschriebene zweiseitige DIN A4 Notiz-/Formelsammlung.
- Verwenden Sie zum Bearbeiten der Aufgaben nur dokumentenechte Schreibgeräte – keinen Bleistift, keine Rotstifte!
- Alle nicht genannten Hilfsmittel sind untersagt. Dies beinhaltet auch jegliche Kommunikation mit Anderen.

Prüfungsdauer

Die Prüfungsdauer beträgt 120 Minuten.

Prüfungsunterlagen

Die Prüfungsunterlagen bestehen aus insgesamt **28** Seiten Aufgabenblättern (dieses Titelblatt, **8** Aufgabenblöcke). Geben Sie immer volle und nachvollziehbare Lösungs- und Rechenwege an.

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 das zusätzliche Lösungsblatt 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.
In den letzten 30 Minuten der Klausur ist eine vorzeitige Abgabe der Klausur nicht möglich. Am Ende der Prüfung bleiben Sie bitte sitzen. Alle Aufgaben- und Lösungsblätter sowie alle zusätzlichen Lösungsblätter sind in die ausgehändigten Umschläge zu geben. Diese werden von der Aufsicht eingesammelt.

Aufgabe	1	2	3	4	5	6	7	8	Σ
≈ Gewichtung ca. [%]	14	12	10	14	11	11	14	14	100
Ergebnis [P]									

Aufgabe 1 Allgemeine Fragen

Beantworten Sie folgende Fragen:

A) Nennen Sie drei verschiedene Beschreibungsformen für Algorithmen.

Pseudo-Code, Programmcode, Ablaufdiagramme, Nassi-Shneiderman-Diagramme, Text

B) Was bedeutet Rekursion bei der Programmierung? Nennen Sie einen Vorteil und einen Nachteil bei der Benutzung.

Funktionen oder Methoden rufen sich selbst wieder auf.

Vereinfacht die Implementierung von manchen Algorithmen. Kann für bessere Lesbarkeit und Verständnis sorgen.

Schlechtere Performanz/Effizienz, Speicherverbrauch auf dem Stack kann sehr groß werden.

C) Nennen Sie 4 Programmierparadigmen und beschreiben Sie kurz, welches Konzept jeweils im Vordergrund steht.

Imperativ: Variablen und Anweisungen als Abstraktion der Prozessor-Funktion

Prozedural (Strukturiert): Aufteilung in verschiedene Unterprogramme (Prozeduren)

Funktional: Mathem. Funktionen weisen Vektoren von Parametern Werte zu

Logikbasiert: Logische Aussagen mit Verifizierung von freien Variablen

Objektorientiert: Objekte interagieren durch Kommunikation miteinander und werden zu Klassen zusammengefasst

Deklarativ: Bestimmt, was programmiert wird

D) Wovon hängt die Größe einer Variablen mit gegebenem Datentyp im Speicher ab? Treffen Sie eine passende Auswahl und geben Sie den Speicherplatz einer Integer und einer Float Variablen an.

Verwendete Prozessorarchitektur.

Bsp.: x86 bzw. 32 Bit Architektur: Integer - 32 Bit, Float – 32 Bit

E) In der folgenden Tabelle sind typische Eigenschaften von Algorithmen aufgelistet. Geben Sie eine kurze Erklärung der Begriffe im Zusammenhang mit Sortieralgorithmen an.

Begriff	**Erklärung**
Effizienz	Gibt an, wie gut die gefundene Lösung im Vergleich mit der optimalen Lösung ist. Performanz/Speicherverbrauch
Komplexität	Gibt an, wie viele Berechnungen abhängig von der Anzahl der Elemente benötigt werden, um eine Lösung zu finden.
Speicherverbrauch	Wie viel Speicher benötigt der Algorithmus zusätzlich zu den zu sortierenden Daten.
Stabilität	Ein Sortieralgorithmus gilt als stabil, wenn bereits bestimmte Positionen eines Elements nicht mehr verändert werden.

F) Nennen und erläutern Sie die drei grundsätzlichen Aufgaben eines Compilers.

Syntaxprüfung: Überprüfung auf gültiges Programm in der Quellsprache

Analyse und Optimierung: Optimierungen, um Performanz, Speicherverbrauch… zu verbessern

Codeerzeugung: Ausführbaren Code (Maschinencode oder Objektdateien) für die Zielarchitektur erzeugen

G) Nennen Sie die beiden Lokalitätsprinzipien in Bezug auf Caches und erläutern Sie diese.

Zeitlich: Benutzte Daten/Befehle werden bald wieder benutzt.

Räumlich: Auf Daten/Befehle, die im Adressraum nahe zu gerade benutzten liegen, wird wahrscheinlicher verwiesen als auf weiter weg liegende.

H) Eine komplexere Datenstruktur mit einer Größe von 1 kB soll an eine Funktion übergeben werden. Dazu stehen die Übergabeprinzipien *Call-by-Value, Call-by-Reference* und *Call-by-Pointer* zur Verfügung. Geben Sie für jedes Prinzip einen Vor- und Nachteil an.

Call-by-Value

Hierbei wird eine Kopie der übergebenen Struktur erzeugt, die innerhalb der Funktion verwendet und bei Beendigung der Funktion wieder zerstört wird.

Vorteil: Die übergebene Struktur wird aufgrund der Kopie nicht verändert. Komplette Trennung der Variablen beim Aufruf

Nachteil: Aufgrund der großen Struktur von 1kB erhöht sich der Speicherverbrauch und die Laufzeit aufgrund der Erzeugung der Kopie.

Call-by-Reference

Hier wird keine Kopie der Struktur erzeugt, sondern lediglich ein anderer Name für die übergebene Struktur innerhalb der Funktion verwendet.

Vorteil: Kein Speicher- und Laufzeit Overhead für die Kopie (nur für Übergabeparameter).

Nachteil: Struktur kann ungewollt überschrieben werden.

Call-by-Pointer

Hier wird die Speicheradresse der Struktur übergeben und keine Kopie erzeugt.

Vorteil: Kein Speicher- und Laufzeit Overhead für die Kopie. Zeiger kann verändert werden.

Nachteil: Umständlicher Zugriff über Pointer (Dereferenzierung); Struktur kann ungewollt überschrieben werden.

Aufgabe 2 C++ Verständnisfragen

A) Was ist das Ergebnis des folgenden C++ Ausdrucks? Dabei ist die Variable *Z* vom Typ **`bool`**.

i. `Z = !(!(5 < 20) ^ (26 % 5));` True [ ] False [X]

ii. `Z = ((5 | 10) - (173 & 1)) == 0;` True [ ] False [X]

B) Durch welche Schlüsselwörter kann die Sichtbarkeit eines Elements in der objektorientierten Programmierung gesteuert werden? (Listen Sie zwei Schlüsselwörter auf)

Public, Private, Protected

C) Angenommen die `int` Variable `i` speichert die Zahl 3, dann werden durch die Befehle

```
cout << "Ausgabe: " << ++i << ",";
cout << i++ << endl;
```

die Bildschirmausgabe

Ausgabe: 4,4

_____

erzeugt.

D) Was berechnet die Funktion **`do()`**?

```
long do(long a)
{
    if (0 < a)
    {
        return do(a - 1) + a;
    }
    return 0;
}
```

Die Funktion bildet die Summe von 0 bis a, 0 + 1 + 2 + … a-1 + a

$$return = \sum_{n=0}^{a} \mathrm{n}$$

E) Welcher Wert steht nach erfolgreichem Schleifendurchlauf in den Variablen **`i,j,k`**?

```
int i, j, k;
for(i = 0, j = 100; i<=j; i++, j--) k = i;
```

i = 51

j = 49

k = 50

F) Welchen Wert speichert der String **s** *nach dem Ausführen der einzelnen Schritte*? Vervollständigen Sie die leeren Zeilen.

```
string s("heute Klausur");
string m("Morgenmantel");

int n = s.find(" ");

s.insert(n, " ");

s.replace(n+3, 3, "Spielkasino", 8, 11);

s.erase (n+6, n+9);

s.replace(0, 6, m, 0, 6);

s.replace(s.find("M"), 1, m, 6, 1);
```

s = “heute Klausur”

s = “heute Kinosur” (zwei Leerzeichen)

s = “heute Kino” (zwei Leerzeichen)

s = “Morgen Kino”

s = “morgen Kino”

G) Gegeben ist der folgende Programmausschnitt. Nehmen Sie an, dass das Programm korrekt und fehlerfrei kompiliert worden ist. Welche Ausgaben werden in der Konsole dargestellt?

```
char n[] = {'a', 'b', 'c'};
char* m = n;
cout << n[0] << " " << *m << endl;

m++;
cout << n[0] << " " << *m << endl;

n[2]++;
cout << n[2] << " " << *m << endl;

*m = '0';
cout << n[1] << " " << *m << endl;
```

a a

a b

d b

0 0

H) 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>
void main()
{
    int v = 3;
    array[5] = {1, 2, 3, 4, 5};
    v = array[5];
    std::cout << "Hallo" << Welt;
    return 0;

}
```

1. void hat kein return Wert oder Funktion muss int zurückgeben
2. array hat kein Bezeichner / Datentyp / Variablentyp
3. Welt ist ein ungültiger Wert / Kommando
4. Zugriff auf array außerhalb gültigen Bereichs

Aufgabe 3 Objektorientierung in C++

Gegeben ist folgendes C++ Programm:

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

class A {
public:
      A() { cout << "A()" << endl; }
      virtual ~A() { cout << "~A()" << endl; }

      void print1() { cout << "A::print1()" << endl; }
      virtual void print2() { cout << "A::print2()" << endl; }
};

class B : public A {
public:
      B() { cout << "B()" << endl; }
      virtual ~B() { cout << "~B()" << endl; }

      void print1() { cout << "B::print1()" << endl; }
      virtual void print2() { cout << "B::print2()" << endl; }
};

class C {
      A a;
      B* b;
public:
      C() { cout << "C()" << endl; b = new B(); }
      ~C() { cout << "~C()" << endl; delete b; }
};


int main() {
      A* a = new A();
      a->print1();
      a->print2();
      {
          B b1;
          b1.print1();
          b1.print2();
      }
      A* b2 = new B();
      b2->print1();
      b2->print2();
      delete b2;
      C c;

      return 0;
}
```

A) Welche Ausgabe liefert das o.g. Programm?

A()
A::print1()
A::print2()

A()
B()

B::print1()
B::print2()

~B()
~A()

A()
B()

A::print1()
B::print2()

~B()
~A()

A()
C()

A()
B()

~C()

~B()
~A()

~A()

B) Erklären Sie kurz die Bedeutung von …

i. Konstruktoren und Destruktoren in Klassen in C++.

Als Konstruktoren und Destruktoren werden in der C++ Programmierung spezielle Prozeduren bzw. Methoden bezeichnet, die beim Erzeugen und Auflösen von Objekten aufgerufen werden.

ii. **`virtual`** als Schlüsselwort für die Methodendeklaration in C++.

Polymorphie: Bei virtuellen Klassenmethoden wird die Einsprungadresse erst zur Laufzeit ermittelt. Dieses sogenannte dynamische Binden ermöglicht es, Klassen von einer Oberklasse abzuleiten und dabei Funktionen zu überschreiben bzw. zu überlagern.

iii. **`private`** als Schüsselwort in Bezug auf Vererbung in C++.

In C++ sind Attribute und Methoden der Zugriffsklasse private ausschließlich in der Klasse verfügbar, in der sie definiert ist.

ODER Private Vererbung: alle Attribute und Methoden werden als private an die Kindklasse vererbt. Public oder Protected Attribute und Methoden sind mit Instanzen der Kindklasse von außen nicht mehr zugreifbar.

C) Benennen und erklären Sie kurz die **vier** Fehler in folgendem C++-Code:

```
class B {
private:
    int value;

public:
    void set_value( int x ) const {
        value = x;
    }
};

class D : public B {
protected:
    int calc() {
        return value * 10;
    }
};

int main() {
    B b;
    D d;
    B& r1 = d;
    D& r2 = b;

    b.set_value( 42 );
    int x = d.calc();

    return 0;
}
```

- **`B::set_value()`** ist als const deklariert und kann daher **`value`** nicht verändern.
- **`B::value`** ist als private deklariert und kann daher nur in **`B`**, aber nicht in **`D`** verwendet werden.
- **`D::calc()`** ist als **`protected`** deklariert und kann daher nicht in **`main()`** aufgerufen werden.
- **`r2`** ist eine Referenz auf **`D`**, daher kann ihr kein Objekt vom Typ **`B`** (Basisklasse von **`D`**) zugewiesen werden.

Aufgabe 4 Datenstrukturen und STL

Eine Universität besitzt einen Veranstaltungskalender, indem alle Vorlesungen, Übungen und Praktika abgespeichert werden. Der Kalender erlaubt es Einträge hinzuzufügen, zu löschen, nach dem Namen zu suchen und den Kalender anzeigen zu lassen.
Der Quellcode der Kalenderklasse ***CVeranstaltungskalender*** ist in Tabelle 4-1 dargestellt. Jede Veranstaltung wird als Objekt der Klasse ***CVeranstaltung*** abgespeichert. Dabei enthält ein Veranstaltungs-Objekt den Namen und den Dozenten der Veranstaltung sowie eine Veranstaltungsnummer.

```
/*Headerdatei: Veranstaltungskalender.h */

#include <list>
#include <string>
#include "Veranstaltung.h"

class CVeranstaltungskalender
{
public:
	CVeranstaltungskalender(void);
	~CVeranstaltungskalender(void);

private:
	std::list<CVeranstaltung *> Veranstaltungen;
	int globaleVeranstaltungsnummer = 0;

public:
	std::string hinzufuegen(std::string name, std::string dozent);
	void loeschen(std::string name);
	void kalender_anzeigen();
	void suche_nach_Dozenten(std::string dozent);
};

/*Headerdatei: Veranstaltung.h */

#include <string>

class CVeranstaltung
{
public:
	CVeranstaltung(void);
	~CVeranstaltung(void);

	CVeranstaltung(int nummer, std::string name, std::string dozent);

private:
	int veranstaltungsnummer;
	std::string name;
	std::string dozent;

public:
	int getVeranstaltungsnummer();
	std::string getName();
	std::string getDozent();
};
```

Tabelle 4-1: Headerdateien des Veranstaltungskalenders

Aufgabe 4.1 Implementierung

Der Veranstaltungskalender soll um eine neue Suchfunktion erweitert werden. In Zukunft soll es möglich sein eine Liste von Veranstaltungen eines Dozenten auszugeben.

A) Implementieren Sie die Funktion ***suche_nach_Dozenten*** um die gewünschte Ausgabe zu erzeugen. Die Funktion erhält als Parameter einen Dozentennamen und erzeugt eine Ausgabe aller Veranstaltungen des Dozenten auf der Konsole. Jede Veranstaltung soll in einer Zeile in folgender Reihenfolge ausgegeben werden: Veranstaltungsnummer, Veranstaltungsname, Dozent.

Zur Implementierung soll ein Iterator explizit verwendet werden. Außerdem soll festgestellt werden, wenn der Dozent keine Veranstaltung hält und falls notwendig soll eine entsprechende Ausgabe auf der Konsole getätigt werden.

```
void CVeranstaltungskalender::suche_nach_Dozenten(std::string dozent)
{
    bool gefunden = false;
    for (list<CVeranstaltung *>::iterator it = Veranstaltungen.begin(); it != Veranstaltungen.end(); ++it)
    {
        if( (*it)->getDozent() == dozent )
        {
            gefunden = true;
            std::cout << (*it)->getVeranstaltungsnummer()  << ", " << (*it)->getName()
            << ", " << (*it)->getDozent() << endl;
        }
    }
    if (gefunden == false)
    {
        std::cout << „Keine Veranstaltungen gefunden“ << endl;
    }
}
```

Aufgabe 4.2 Allgemeine Fragen zur STL

A) Die STL Container lassen sich in drei Typen einteilen. Geben Sie den Container-Typ an, der in der Headerdatei des Veranstaltungskalenders zum Speichern der Veranstaltungen verwendet wird:

List ist ein Sequentieller Container der STL

B) Nennen Sie zwei weitere Container-Typen der STL:

Assoziative Container

Container Adapter

C) Implementieren Sie einen alternativen STL-Container mit dem das Suchen und das Sortieren der Veranstaltungen nach der Veranstaltungsnummer vereinfacht werden. Geben Sie außerdem eine kurze Begründung an.

map <int /*Veranstaltungsnummer*/, CVeranstaltung>

Eine Map verwendet Hash-Keys um die Elemente abzuspeichern. Dabei werden Sie nach dem Key sortiert. Das Verwenden der Veranstaltungsnummer als Key ermöglicht dabei eine automatische Sortierung und eine effiziente Suche.

Aufgabe 4.3 Nassi-Shneiderman-Diagramm

Die Funktion ***hinzufügen*** der Klasse ***CVeranstaltungskalender*** übernimmt die folgende Funktion:

Bei Aufruf werden ein Veranstaltungsname und der Dozent der Veranstaltung übergeben. Es wird überprüft, ob eine Veranstaltung mit gleichem Namen schon in der Liste der Veranstaltungen enthalten ist. Wenn eine entsprechende Veranstaltung schon vorhanden ist, dann wird der Text „Veranstaltung schon eingetragen.“ zurückgegeben. Ansonsten wird ein neues Veranstaltungsobjekt mit einer neuen Veranstaltungsnummer, dem Namen und dem Dozenten erzeugt und „Veranstaltung hinzugefügt.“ ausgegeben.

A) Stellen Sie den Funktionsablauf in einem Nassi-Shneiderman-Diagramm mit Pseudo-Code dar:

<table>
<tr><td colspan="2">Eingabe: Name, Dozent</td></tr>
<tr><td colspan="2">Gefunden = nein</td></tr>
<tr><td colspan="2">i = 0</td></tr>
<tr><td colspan="2">Solange i < Listengröße UND gefunden = nein
Name == Name_der_Veranstaltung[i]?
Gefunden = ja | %</td></tr>
<tr><td colspan="2">Gefunden == ja?</td></tr>
<tr><td>Return: „Veranstaltung schon vorhanden.“</td><td>Globale Veranstaltungsnummer ++</td></tr>
<tr><td rowspan="3">%</td><td>Erzeuge v = Veranstaltung(GlobaleVeranstaltungsnummer, Name, Dozent)</td></tr>
<tr><td>Füge Veranstaltung v zur Liste hinzu</td></tr>
<tr><td>Return: „Veranstaltung hinzugefügt.“</td></tr>
</table>

Aufgabe 5 Datei-/Stream-/Stringverarbeitung in C++

Es soll eine C++-Klasse **`LaenderListe`** implementiert werden, die eine Liste von Ländern verwaltet. Die Liste soll aus einer Datei eingelesen werden, die in jeder Zeile Daten über jeweils ein Land enthält. Jede Zeile besteht aus dem Ländernamen, dem Kontinent und der Einwohnerzahl, die jeweils durch ein Semikolon getrennt sind (siehe Abbildung 5-1).

```
Deutschland;Europa;80493000
Vereinigte Staaten von Amerika;Nordamerika;314167157
Österreich;Europa;8488511
Kanada;Nordamerika;35056064
Brasilien;Südamerika;192380000
Japan;Asien;126659683
```

Abbildung 5-1: Beispieldaten

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

class Land {
public:
    Land(string name, string kontinent, double einwohner);
    string name() const;
    string kontinent() const;
    double einwohner() const;
private: // ...
};

class LaenderListe {
public:
    LaenderListe();
    bool einlesen(string name);
    bool ausgeben(string name);
private:
    void landEinfuegen(string zeile);
    static string ersetzen(string name, string alt, string neu);
    static string umlauteEntfernen(string name);
    vector<Land> laender;
};
```

A) Implementieren Sie die Methode **`einlesen()`**. Sie soll eine Datei zeilenweise einlesen. Der Pfad zu der Datei wird der Funktion über den Parameter **`name`** übergeben. Jede eingelesene Zeile soll mit Hilfe der Methode **`landEinfuegen()`** in die Datenbank eingefügt werden. Falls die Eingabedatei nicht geöffnet werden kann, soll die Funktion den Wert **`false`** zurückgeben, bei erfolgreicher Verarbeitung der Eingabedatei soll sie den Wert **`true`** zurückgeben.

```
bool LaenderListe::einlesen( string name )
{

    fstream infile(name, ios::in);  // alt.: ifstream ohne ios::in
    if (!infile) {  // alt.: infile.good(), !infile.fail(), etc.
        return false;
    }
    string line;
    while (!infile.eof()) {  // alt.: while(getline(…)), infile.good(),…
        getline(infile, line);
        landEinfuegen(line);
    }
    return true;









}
```

B) Die Methode **`ersetzen()`** wird in der Methode **`umlauteEntfernen()`** verwendet, um alle Umlaute aus einem String zu entfernen.
Implementieren Sie die Methode **`ersetzen()`**. Sie soll im String, der über den Parameter **`name`** übergeben wird, **alle** Vorkommnisse des über den Parameter **`alt`** übergebenen Strings durch den über den Parameter **`neu`** übergebenen String ersetzen. Der Rückgabewert der Funktion soll der modifizierte String sein.
Hinweis: Beachten Sie, dass der zu ersetzende String aus mehreren Zeichen bestehen kann.

```
string LaenderListe::ersetzen( string name, string alt, string neu )
{

    string::size_type pos;          //alt. int pos;
    while ((pos = name.find(alt)) != string::npos) { // alt. != -1
        name.replace(pos, alt.length(), neu);
        // alt. name.erase() und name.insert()
    }
    return name;









}
```

C) Die Methode **`landEinfuegen()`** erwartet als Eingabeparameter **`zeile`** jeweils eine Zeile der in Abbildung 5-1 dargestellten Eingabedatei. Diesen String zerlegt sie in ihre jeweils durch ein Semikolon getrennten Einzelteile und legt sie in den Variablen **`name`**, **`kontinent`** und **`einwohner`** ab. Hierbei dürfen die den Variablen **`name`** und **`kontinent`** zugewiesenen Strings keine Umlaute enthalten.
Abschließend wird ein neues Objekt vom Typ **`Land`** erzeugt, und im Vektor **`laender`** abgelegt. Hierbei werden die zuvor angelegten Variablen dem Konstruktor von **`Land`** als Parameter übergeben.
Vervollständigen Sie die Implementierung der Methode **`landEinfuegen()`**.

Hinweise:
- In jedem Schritt soll zunächst das Semikolon gesucht werden. Falls es gefunden wurde, soll der Teilstring vor dem Semikolon in der entsprechenden Variablen abgelegt werden. Bevor nach dem nächsten Semikolon gesucht wird, soll in der Variable **`zeile`** der Teilstring, der auf das bereits gefundene Semikolon folgt, abgelegt werden.
- Verwenden Sie für die Entfernung der Umlaute die Methode **`umlauteEntfernen()`**, die einen String als Parameter erwartet und einen umlautfreien String zurückgibt.
- Beachten Sie, dass die Variable **`einwohner`** den Typ **`double`** hat.

```
void LaenderListe::landEinfuegen( string zeile )
{
    string::size_type pos;
    pos = A;
    if (pos != B) {
        string name = C;
        zeile = zeile.substr(D);
        pos = A;
        if (pos != B) {
            string kontinent = C;
            double einwohner = E;
            laender.push_back( Land(name, kontinent, einwohner) );
        }
    }
}
```

A zeile.find(";")

B string::npos oder -1

C umlauteEntfernen(zeile.substr(0, pos))

D pos+1

E atof(zeile.substr(pos+1).c_str()) // alt. atoi()

Aufgabe 6 Graphen und Algorithmen

Aufgabe 6.1 Graphentheorie

Gegeben sei der folgende Graph:

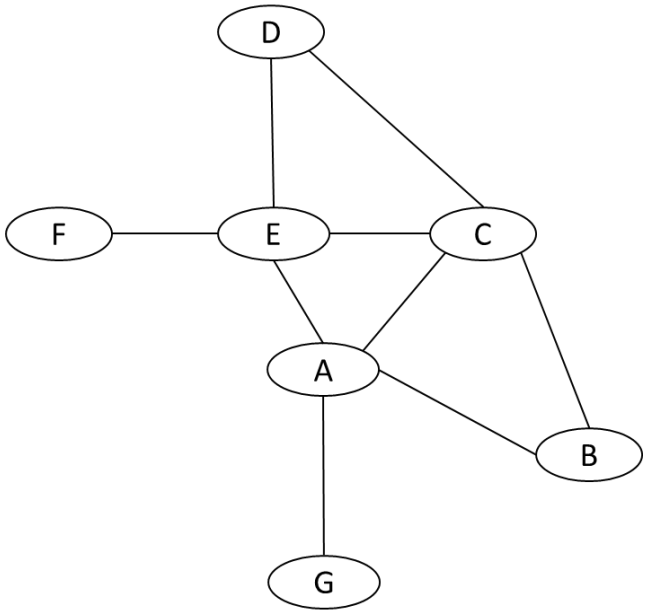


Abbildung 6-1: Graph

A) Wie viele Knoten und Kanten hat dieser Graph?

Knoten: 7

Kanten: 9

B) Existiert ein Zyklus der Länge 4 in dem gegebenen Graphen? Falls ja, geben Sie ihn an. Falls nein, geben Sie die maximale Länge der vorhandenden Zyklen an.

Ja.

A-C-D-E bzw. A-B-C-E.

C) Bestimmen Sie den maximalen Knotengrad im gegebenen Graphen. Geben Sie alle Knoten mit diesem Grad an.

4.

A, C, E.

D) Stellen Sie den gegebenen Graphen in Form einer Adjazenzmatrix dar.

	A	B	C	D	E	F	G
A	0	1	1	0	1	0	1
B	1	0	1	0	0	0	0
C	1	1	0	1	1	0	0
D	0	0	1	0	1	0	0
E	1	0	1	1	0	1	0
F	0	0	0	0	1	0	0
G	1	0	0	0	0	0	0

E) Diese Adjazenzmatrix hat zwei spezielle Eigenschaften. Geben Sie **eine** an und begründen Sie sie.

1) Die Adjazenzmatrix ist symmetrisch über die Hauptdiagonale.

Da der Graph ungerichtet ist, existiert eine Kante von B nach A, wenn eine Kante von A nach B existiert.

2) Die Adjazenzmatrix ist spurfrei (Hauptdiagonale ist 0)

Alle Knoten sind nicht reflexiv (haben keine Kante auf sich selbst)

Aufgabe 6.2 Tiefensuche

Der gegebene Graph aus dem ersten Teil wurde mit Information über die Abstände zwischen den Knoten erweitert:

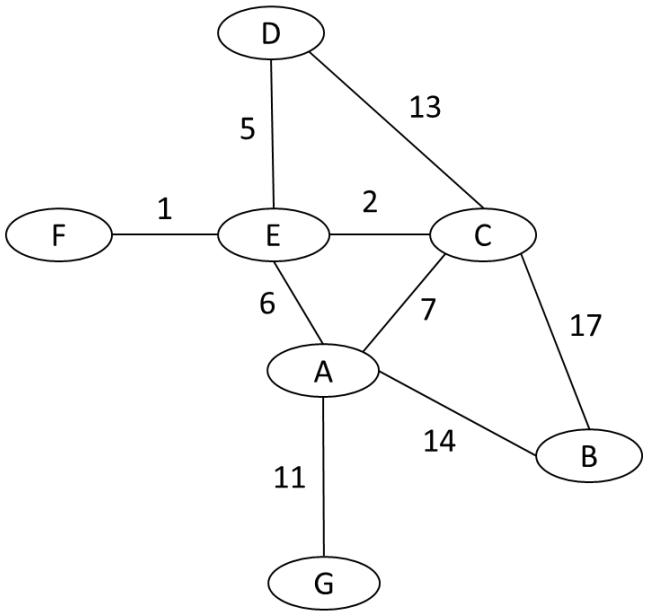


Abbildung 6-2: Erweiterter Graph mit Knotenabständen

Außerdem beschreibt der folgende Pseudocode den Tiefensuche-Algorithmus:

```
DepthFirstSearch( G )
  for alle Knoten u in G
  do farbe[u] = weiss
     vater[u] = NIL
  zeit = 0
  for alle Knoten u in G
  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
```

A) Welche Rolle spielt das Array **`farbe[]`** in dem Algorithmus? Beschreiben Sie die Bedeutung der benutzten Farben.

Das Array speichert den Bearbeitungszustand jedes Knotens an der entsprechenden Stelle.

Weiss: Der Knoten wurde noch nicht besucht.

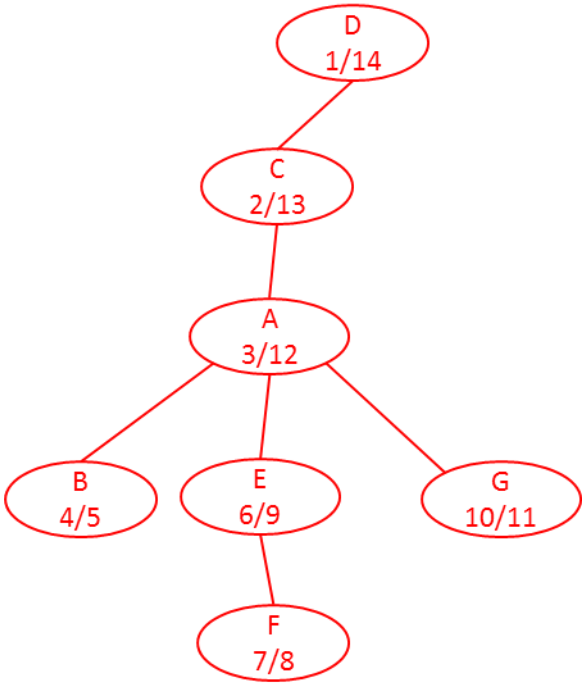
Grau: Der Knoten wird aktuell betrachtet.

Schwarz: Die Bearbeitung des Knoten ist abgeschlossen.

Wenden Sie den beschriebenen Tiefensuche-Algorithmus auf den gegebenen Graphen an. Beginnen Sie mit dem Knoten D. Betrachten Sie die Nachfolgerknoten nach der alphabetischen Reihenfolge.

B) Entsteht bei der Suche ein Baum oder ein Tiefensuchwald? Zeichnen Sie ihre Lösung samt Entdeckungs- und Endzeiten aller Knoten.

Es entsteht ein Baum.



C) Geben Sie den dabei gefundenen Weg von Knoten D nach Knoten B an. Berechnen Sie dessen Länge.

D-C-A-B

Länge = 13 + 7 + 14 = 34

D) Nun sollte der Nachfolgerknoten nicht alphabetisch, sondern nach dem kürzesten Abstand zum Vorgängerknoten bestimmt werden. Ist der so modifizierte Algorithmus prinzipiell für die Bestimmung des kürzesten Weges von Knoten X zu Knoten Y geeignet? Begründen Sie Ihre Antwort.

Nein, er ist nicht geeignet.

Alternative Begründung:

1) Nach dem Algorithmus wird höchstens ein Weg von X nach Y betrachtet/gefunden. Die Suche nach dem kürzesten Weg erfordert aber, dass alle möglichen Wege betrachtet werden.
2) Gegenbeispiel zeichnen.
3) Wenn Wege direkt aus der Reihenfolge der besuchten Knoten bestimmt werden, können sie Schleifen enthalten (z.B. D-C-A-B-A-E statt D-C-A-E). Wenn die Summe aller Kantengewichte der Schleife (2x A-B) größer als der direkte Abstand (A-E) ist, ist der gefundene Weg nicht der kürzeste.

Aufgabe 7 Algorithmusanalyse

Gegeben ist der C++ Code eines angepassten Swap-Sort Algorithmus. Dieser sortiert ein Array in aufsteigender Reihenfolge. Der gezeigten Funktion wird ein Pointer auf ein Array **`sortMe`** und die Anzahl der Elemente **`length`** übergeben.

```
01  void swapSort (int* sortMe, int length) {
02    int sortval = 0;
03    bool* markerarr = new bool[length];
04    for (int i = 0; i < length; i++) {
05      markerarr[i] = false;
06    }
07
08    while (sortval < length) {
09      int counter = 0;
10      for (int i = 0; i < length; i++) {
11        if (sortMe[i] < sortMe[sortval]) {
12          counter++;
13        }
14      }
15
16      int tmp = sortMe[sortval];
17      sortMe[sortval] = sortMe[counter];
18      sortMe[counter] = tmp;
19      markerarr[counter] = true;
20
21      sortval = 0;
22      while (markerarr[sortval] == true) {
23        sortval++;
24      }
25    }
26  }
```

Aufgabe 7.1 Algorithmenverständnis

A) Erklären Sie die Aufgabe der Zeilen 09-14 innerhalb des Algorithmus. Was ist nach dem Ende der Schleife der Inhalt in der Variablen **`counter`** in Bezug auf das Array?

In den Zeilen 09-14 wird gezählt wie viele Elemente innerhalb des gesamten Arrays kleiner sind als der Wert an der Position **`sortval`** des Arrays. Diese Anzahl wird in der Variablen **`counter`** gespeichert (und ist somit die Position an welcher der Wert in einem sortierten Array stehen muss).

B) Welche Aufgabe hat innerhalb des Algorithmus das Array **`markerarr`**?

Das Array **`markerarr`** speichert an welchen Stellen des Arrays bereits sortierte Werte stehen (und überwacht somit, dass ein sortiert Wert nicht nochmals bearbeitet wird).

Aufgabe 7.2 Algorithmenanalyse

A) Der Funktion wird das folgende Array {4, 2, 8, 6} mit der Länge 4 übergeben. Ermitteln Sie die Ausführung des Algorithmus und füllen Sie jeweils eine Zeile in der folgenden Tabelle aus, immer wenn der Algorithmus die Zeile 15 und 20 erreicht. Verwenden Sie dabei innerhalb des **`markerarr`** 0 für **`false`** und 1 für **`true`**.

				Array sortMe				Array markerarr			
		Index des Arrays →		0	1	2	3	0	1	2	3
Zeile	sortval	counter		4	2	8	6	0	0	0	0
15	0	1		4	2	8	6	0	0	0	0
20	0	1		2	4	8	6	0	1	0	0
15	0	0		2	4	8	6	0	1	0	0
20	0	0		2	4	8	6	1	1	0	0
15	2	3		2	4	8	6	1	1	0	0
20	2	3		2	4	6	8	1	1	0	1
15	2	2		2	4	6	8	1	1	0	1
20	2	2		2	4	6	8	1	1	1	1

B) Führen Sie zum gezeigten Code eine Laufzeitanalyse durch und füllen Sie dabei für die Zeilen 2-11 die folgende Tabelle für den Worst-Case Fall aus. Geben Sie jeweils die Anzahl der Ausführungen in Abhängigkeit von **n**, welches der Anzahl der Elemente im Array entspricht, an (für das Beispiel in Aufgabenteil A) wäre n = 4).

Zeile		Anzahl der Ausführungen
02 03	`int sortval = 0;` `bool* markerarr = new bool[length];`	1
04	`for (int i = 0; i < length; i++) {`	n+1
05 06	`  markerarr[i] = false;` `}`	n
08	`while (sortval < length) {`	n+1
09	`  int counter = 0;`	n
10	`  for (int i = 0; i < length; i++) {`	n(n+1)
11	`  if (sortMe[i] < sortMe[sortval]) {`	n^2

Aufgabe 7.3 Algorithmenentwurf

A) Der Algorithmus soll nun das folgende Array mit der Länge 5 sortieren:

```
int x[5] = { 5, 6, 6, 3, 7 }.
```

Was passiert während des Programmablaufs. Beschreiben Sie den Fehler **und** die Ursache.

Das Programm wird entweder abstürzten oder in der Endlosschleife ausgeführt. Die Ursache des Problems besteht darin, dass innerhalb des Arrays eine Zahl mehrmals vorkommt. Dies wird beim Sortieren nicht bemerkt und die Zahl immer wieder mit der anderen gleichen Zahl getauscht.

B) Wie muss das Programm verändert bzw. ergänzt werden, damit der Algorithmus für beliebige Arrays korrekt arbeitet. Sie dürfen das Programm nur innerhalb der Zeilen 09-24 verändern bzw. ergänzen. Sie können die Lösung entweder textuell beschreiben oder direkt Programmcode angeben.

Hinweis: Geben Sie für Veränderungen bzw. Ergänzungen immer die entsprechende Zeilennummer des Quellcodes an.

Das Programm muss erkennen, dass an der Position mit welcher es die Zahl tauschen will (Zeile 15), bereits schon vorher für einen Tausch verwendet wurde. Dies kann direkt über das **markerarray** erkannt werden, da an der entsprechenden Stelle ein **true** steht. In diesem Fall wird einfach die nächste Position im Array verwendet bzw. wenn diese auch schon verwendet wurde, wiederum die nächste Stelle usw.

Zeile 15: `for (;markerarr[counter] == true; counter++) { }`

`if (markerarr[counter] == true) {counter++}` funktioniert nur wenn die Zahl zweimal und nicht mehrmals vorkommt

Aufgabe 8 Rechnerarchitektur

Aufgabe 8.1 Allgemeine Fragen zur Rechnerarchitektur

A) Erläutern Sie die folgenden vier Adressierungsmodi:

Register mode: Der Operand ist in einem der Register gespeichert.

Direct mode: Die Speicher-Adresse des Operanden ist in der Instruktion gespeichert.

Immediate mode: Der Operand ist Teil der Instruktion.

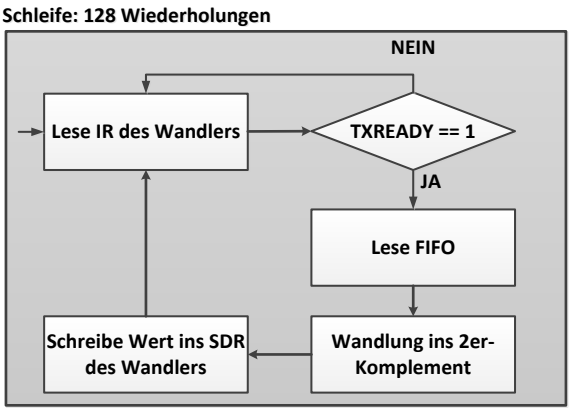
Autoincrement mode: Die Speicher-Adresse des Operanden ist in einem der Register gespeichert. Der Inhalt dieses Registers wird automatisch inkrementiert nachdem die Adresse gelesen wurde.

B) Nennen Sie einen Vorteil und einen Nachteil der Cacheorganisation n-Weg assoziativ gegenüber direkt-abbildend.

n-Weg assoziativ hat den Nachteil, dass man n-1 mehr Komparatoren als direkt-abbildend braucht, hat aber den Vorteil der höheren Flexibilität beim Mappen von Daten aus dem Hauptspeicher in das Cache und dadurch eine höhere cache-hit-rate (jedes Byte kann in n verschiedenen cache-sets gemappt werden).

Aufgabe 8.2 Hardwarenahe Programmierung

Schreiben Sie eine C++ Codesequenz, die 128 Werte mit 32 Bit Breite aus einem FIFO ausliest und an einen DAC-Wandler übergibt. Das nebenstehende Flussdiagramm stellt den Ablauf der Sequenz dar.



Der Wandler kann nur dann einen neuen Wert verarbeiten, wenn er das Bit 29 (TXREADY) im Interrupt-Register (IR) auf 1 gesetzt hat. Ist der Wandler bereit, kann der Wert aus dem FIFO ausgelesen und in das Sample-Data-Register (SDR) des Wandlers geschrieben werden.

Die Daten aus dem FIFO müssen vor dem Schreiben ins SDR einer Zweierkomplement-Wandlung unterzogen werden (siehe Flussdiagramm).

Hinweise: Die Adresse der beiden Geräte und die Offsets der benötigten Register sind im zu ergänzenden Code-Abschnitt gegeben. Das Interrupt-Register wird automatisch auf Null gesetzt, wenn es gelesen wird. Die Bitreihenfolge in den Registern ist 31..0 (von links nach rechts).

```
uint32_t*   dac_basis = 0xFFFF4000; // Adresse des DAC-Wandlers
uint32_t    sdr_off = 0x00000000;   // Offset Sample-Data Register
uint32_t    ir_off = 0x0000001C;    // Offset Interrupt Register
uint32_t*   fifo = 0x10000000;      // Adresse des FIFOs
uint32_t    temp;                   // Hilfsvariable

// fügen Sie hier Ihren C++ Code ein:

for ( int i = 0; i < 128; i++ ) {
    do {
        temp = *(dac_basis+ir_off);
        temp = temp & 0x20000000;
    } while ( temp == 0 );


    temp = *fifo;

    temp = (~temp) + 1;

    *(dac_basis+sdr_off) = temp;
}
```

Aufgabe 8.3 Cache

Ein Prozessor adressiert seinen Hauptspeicher mit 32 Bit. Es soll nun ein Cachebaustein in SRAM Technologie integriert werden, der 1024 KByte Daten aus dem Hauptspeicher aufnehmen kann. Der Cache ist als 2-Weg assoziativ Cache ausgeführt und kann pro Cacheadresse 128 Bit Daten aus dem Hauptspeicher aufnehmen.

Bestimmen Sie die Speicherorganisation des Caches bezüglich der im Cache abzulegenden Daten und der Adressierung, indem Sie die 13 offenen Felder in der folgenden Grafik ausfüllen und die 4 offenen gestrichelten Verbindungen ergänzen. **Berechnen Sie zunächst die Größe des Offsets, des Index und des Tags in Bit** und tragen Sie dann in die gepunkteten Rechtecke die passenden Zahlen und in die gestrichelten Rechtecke die passenden Begriffe bzw. Symbole ein.

Offset = ld(Blockgröße) = ld(128 / 8) = ld(16) = 4 Bit
Index = ld(Blöcke/Sets) = ld((1024*1024/16) / 2) = ld(32768) = 15 Bit
Tag = Adressbreite - Offset - Index = 32 – 4 – 15 = 13 Bit