
Aufgaben zur Klausur **C** und **Objektorientierte Programmierung** im SS 97 (WI63)

Zeit: 150 Minuten

erlaubte Hilfsmittel: keine

Bitte tragen Sie Ihre Antworten und fertigen Lösungen ausschließlich an den gekennzeichneten Stellen in das Aufgabenblatt ein. Ist ihre Lösung wesentlich umfangreicher, so überprüfen Sie bitte nochmals Ihren Lösungsweg.

Sollten Unklarheiten oder Mehrdeutigkeiten bei der Aufgabenstellung auftreten, so notieren Sie bitte, wie Sie die Aufgabe interpretiert haben.

Bei allen C Routinen sollen die Parameter und Funktionsresultate mit ANSI-C Prototypen angegeben werden.

Viel Erfolg !

Diese Klausur besteht einschließlich dieses Deckblattes aus 11 Seiten

Aufgabe 1:

Gegeben sei das folgende C-Programm. Hinweis: der \ll Operator hat geringere Priorität als + und -.

```
#include <stdio.h>
#include <math.h>

#define twotimes(x) x + x
#define sqr(x) x * x
#define min(x,y) x < y ? x : y
#define squareroot(x) (++cnt, sqrt(x))

int cnt = 0;

int main(void)
{
    int i = 4, j = 3, r;
    double f = 1.25;

    r = twotimes(i << 2);
    printf(".1) r = %d\n",r);

    r = sqr(i + j);
    printf(".2) r = %d\n",r);

    r = 2 * twotimes(i);
    printf(".3) r = %d\n",r);

    r = sqr(twotimes(i));
    printf(".4) r = %d\n",r);

    r = twotimes(2*f);
    printf(".5) r = %d\n",r);

    r = min(twotimes(i+1),sqr(i));
    printf(".6) r = %d\n",r);

    r = min(squareroot(i+1),squareroot(j+1));
    printf(".7) r = %d, cnt = %d\n",r,cnt);

    return 0;
}
```

Welche 7 Ausgabezeilen erzeugt dieses Programm

.....

.....

.....

.....

.....

.....

.....

Ändern Sie das Makro *sqr* so ab, daß es für .2) den gewünschten Effekt hat.

.....

Ändern Sie das Makro *twotimes* so ab, daß es sich immer wie eine Funktion verhält

.....

Warum kann das Makro *min* nicht so verändert werden, das es sich immer, bei jedem Gebrauch, wie eine gleichwertige Funktion verhält?

.....

Warum kann der Gebrauch von Makros in der Form von *min* die Programmausführung verlangsamen (im Vergleich zu Funktionen).

.....

Aufgabe 2:

Gegeben sei das folgende C-Programm:

```
#include <stdio.h>

int main(void)
{
    int i = 5;

    printf("%d,%d\n",i++,i++);

    return 0;
}
```

Dieses Programm gibt eine Zeile aus. Welche Ausgaben sind für dieses Programm vom ANSI-C Standard zugelassen:

.....

.....

.....

.....

.....

Aufgabe 3:

Die folgende C-Funktion testet, ob die Zeichenreihe $s1$ Anfangsstück der Zeichenreihe $s2$ ist.

```
int isPrefix(char * s1, char * s2)
{
    return
        *s1 == 0
        || ( *s1 == *s2
            &&
            isPrefix(s1+1, s2+1) );
}
```

Formen sie diese Funktion um in eine gleichwertige, die keine Funktionsaufrufe mehr enthält.

.....

.....

.....

.....

.....

.....

.....

.....

.....

Aufgabe 4:

Welche der folgenden Ausdrücke sind equivalent zu dem Ausdruck $a[j][k]$

$$*(a[j] + k)$$

ja nein

$$** (a[j + k])$$

ja nein

$$*(a + j)[k]$$

ja nein

$$*(a + k)[j]$$

ja nein

$$*((*(a + j)) + k)$$

ja nein

$$** (a + j) + k$$

ja nein

$$*(&a[0][0] + j + k)$$

ja nein

Aufgabe 5:

Gegeben sei das folgende Programm, das mit virtuellen Funktionen arbeitet:

```
#include <iostream.h>

int linecnt = 0;

class X { public:
    virtual const char * f1 () const { return " X"; }
    virtual const char * f2 () const { return " X"; }
    virtual void g () const {
        cout << ++linecnt << "." << f1() << f2() << endl;
    }
};

class Y : public X { public:
    virtual const char * f1 () const { return " Y"; }
};

class Z : public Y { public:
    virtual const char * f2 () const { return " Z"; }
};

int main() {
    Y y1; y1.g();
    Z z1; z1.g();

    X x1(y1); x1.g();
    Y y2(z1); y2.g();

    X & x3 = z1; x3.g();
    X & x4 = y1; x4.g();

    X * p2 = new Y(); p2->g();
    Z * p3 = new Z(); p3->g();

    X * p4 = p3; p4->g();

    return 0;
}
```

Welche 9 Ausgabezeilen erzeugt dieses Programm

.....

.....

.....

.....

.....

.....

.....

.....

.....



Aufgabe 6:

Gegeben ist das folgende C++ Programmstück. Dieses soll in eine header Datei *p.h* und eine Implementierungsdatei *p.cc* aufgeteilt werden, so daß es von mehreren anderen Modulen verwendet werden kann. Kennzeichnen Sie durch Ankreuzen des *.h* Feldes für die header Datei oder *.cc* für die Implementierungsdatei, in welche Datei die einzelnen Codestücke übertragen werden müssen.

```
static int counter; //  .h  .cc

typedef unsigned long Nat; //  .h  .cc

class X; //  .h  .cc

const Nat limit = 8888; //  .h  .cc

extern Nat theAnswer; //  .h  .cc

#include <iostream.h> //  .h  .cc

static int counter = 4711; //  .h  .cc

static int foo(const X & x1); //  .h  .cc

int bar(const X & x1); //  .h  .cc

Nat theAnswer = 42; //  .h  .cc

class X { //  .h  .cc
private:
    Nat d;
    int check() const;
public:
    X();
    X(const & Nat);
    Nat get() const;
    friend ostream & operator >> (ostream &, const X &);
};

Nat X :: get() const { //  .h  .cc
    return d + counter;
}

#define lowerbound 1111 //  .h  .cc

int X :: check() const { //  .h  .cc
```

```

    return d ≥ lowerbound;
}

ostream & operator >> (ostream & o, const X & x1) { // .h .cc
    return
        o << x1.d << endl;
}

extern Nat numbers[]; // .h .cc

int bar(const X & x1) { // .h .cc
    return
        foo(x1) + 2;
}

static int foo(const X & x1) { // .h .cc
    return
        x1.get();
}

inline X::X () { d = 0; } // .h .cc

struct Field { int data[lowerbound]; }; // .h .cc

extern struct Field aField; // .h .cc

```

Aufgabe 7:

In der Objektorientierten Programmierung spielt die Wiederverwendung eine zentrale Rolle. Dabei können schon entwickelte, fertige Klassen von anderen Klassen beerbt oder benutzt werden.

Vererbung

```
class Y : public X { ... };
```

Benutzung

```
class Y {  
    X d;  
    ...  
};
```

Welches sind Vorteile der Vererbung gegenüber der Benutzung der Klasse X in der Klasse Y :

- | | | | |
|--|-----------------------------|-------------------------------|-------------------------------------|
| 1. Der Quellcode wird kürzer. | ja <input type="checkbox"/> | nein <input type="checkbox"/> | weiß nicht <input type="checkbox"/> |
| 2. Der Objektcode wird kürzer. | ja <input type="checkbox"/> | nein <input type="checkbox"/> | weiß nicht <input type="checkbox"/> |
| 3. Die Laufzeit des Programms verringert sich. | ja <input type="checkbox"/> | nein <input type="checkbox"/> | weiß nicht <input type="checkbox"/> |
| 4. Die Compilierzeit verringert sich. | ja <input type="checkbox"/> | nein <input type="checkbox"/> | weiß nicht <input type="checkbox"/> |
| 5. Der Quelltext wird selbstdokumentierend. | ja <input type="checkbox"/> | nein <input type="checkbox"/> | weiß nicht <input type="checkbox"/> |
| 6. Die Erweiterbarkeit ist verbessert. | ja <input type="checkbox"/> | nein <input type="checkbox"/> | weiß nicht <input type="checkbox"/> |
| 7. Die Veränderung der Klasse X hat keinen Einfluß auf Programmcode außerhalb von Y . | ja <input type="checkbox"/> | nein <input type="checkbox"/> | weiß nicht <input type="checkbox"/> |
| 8. Die Namensgebung für Methoden in Y kann besser an die Aufgabenstellung angepaßt werden. | ja <input type="checkbox"/> | nein <input type="checkbox"/> | weiß nicht <input type="checkbox"/> |
| 9. Die Konstruktoren brauchen nicht neu entwickelt werden. | ja <input type="checkbox"/> | nein <input type="checkbox"/> | weiß nicht <input type="checkbox"/> |
| 10. Die Speicherverwaltung wird einfacher. | ja <input type="checkbox"/> | nein <input type="checkbox"/> | weiß nicht <input type="checkbox"/> |
| 11. Die modulare Verständlichkeit nimmt zu. | ja <input type="checkbox"/> | nein <input type="checkbox"/> | weiß nicht <input type="checkbox"/> |
-