Prüfung C/C++ (10.01.2005)

Vorname ....

Nachname ....

Matrikelnummer ....

Allgemeine Hinweise

Die Fragen sind durch freien (*aber bitte lesbaren*) Text im entsprechenden Feld zu beantworten. Der vorgesehene Platz ist **NICHT** bindend: Einerseits kann man manche Fragen auch sinnvoll beantworten, ohne den ganzen Platz zu benötigen; andererseits kann man auch auf einem zusätzlichen Blatt fortfahren, wenn man einen Roman schreiben will. Dann aber die entsprechende Antwort mit der Nummer der Frage kennzeichnen!

Bitte nur mit dokumentenechten Stiften schreiben (**kein Bleistift!**), sowie nicht mit roter (oder ähnlicher) Farbe.

Nicht deutlich lesbare Antworten werden nicht gewertet!

Jedes zusätzliche Blatt bitte mit Name und Matrikelnummer versehen, und spätestens bei der Abgabe mit den ausgeteilten Blättern zusammenheften.

Im Notfall fragen

Wenn eine Frage unklar oder falsch formuliert ist, dann lieber sofort fragen, anstatt Zeit zu verschwenden.

Wer eine Wissenslücke hat, die man schnell beheben könnte, und wegen dieser Lücke den Rest einer Aufgabe nicht lösen kann, kann sich einen Hinweis geben lassen, der mit dem Anschreiben angemessener Strafpunkte einhergeht. Damit kann er dann hoffentlich den Rest der Aufgabe lösen und Folgefehler vermeiden.

Und jetzt bitte beruhigen, tief Luft holen, und viel Glück!


PERSCHEID
www.martin-perscheid.de
SCHNELL WURDEN UNS DIE GRENZEN DER DIGITALISIERUNG BEWUSST, ALS ALICE VERSUCHTE, SICH LASZIV AUF DEM KEYBOARD ZU RÄKELN.

Prüfungsfragen

1. In ANSI-C soll eine einfach verkettete Liste irgendwelcher Nutzdaten aufgebaut werden (nullterminierte Zeichenketten).

   Jeweils ein Listenelement soll in einer `struct` der Art:

   ```
   typedef struct Listenelement
   {
       char                    nutzdaten[100];
       struct Listenelement  * next;
   } Listenelement_t;
   ```

   abgelegt werden.

   Der Aufrufer soll einen Zeiger auf das erste Listenelement als automatische Variable haben:

   ```
   int main()
   {
      struct Listenelement  * listenanfang = NULL;
      // ...
   }
   ```

   Aufgabe: Es soll eine Funktion `einfanf()` angegeben werden, welche

   - in einer geeigneten Art und Weise den Listenanfang übergeben bekommt (in der Parameterliste),
   - eine nullterminierte Zeichenkette (die von der Funktion nicht verändert wird) übergeben bekommt,
   - die übergebene Zeichenkette als neues Element an den Anfang der Liste einfügt,
   - als Rückgabewert liefert, ob das Einfügen geklappt hat

   Die Funktion soll auch bei einer leeren Liste erfolgreich einfügen!

   ....
   ....
   ....
   ....
   ....
   ....
   ....
   ....
   ....
   ....

2. Wie sieht für die vorige Aufgabe der Aufruf der Funktion `einfanf()` aus (bitte mit Prüfung des Rückgabewerts!)?

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

3. Bitte beschreiben Sie nachvollziehbar, was beim Aufruf Ihrer Funktion `einfanf()` aus den vorigen beiden Aufgaben passiert...

   (a) ...für den Fall, daß die Liste beim Aufruf noch leer ist:

(b) ...für den Fall, daß die Liste beim Aufruf bereits zwei Elemente enthält:

4. Es soll ein `template` erstellt werden, welches ein Feld von Werten beliebigen Typs repäsentiert (mit einem Index ab 0).

   Im Konstruktor soll beim Anlegen die Anzahl der Elemente angegeben werden.

   Bitte ergänzen Sie den folgenden Quelltext entsprechend, sodaß die anschließend gezeigte Verwendung sinnvoll möglich ist:

```
// Time-stamp: "09.01.05 18:33 tarr.cpp klaus@wachtler.de"

#include <iostream>

                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
                                                       // auszufüllen
```

```
int main( int nargs, char **args )
{

  Arr<int>   einFeld( 10 );

  einFeld[0] = 10;
  einFeld[5] = 5;
  einFeld[9] = einFeld[0] + einFeld[9];

  Arr<double>   einanderesFeld( 10 );

  einanderesFeld[0] = 10.5;
  einanderesFeld[5] = 5.9;
  einanderesFeld[9] = einanderesFeld[0] + einanderesFeld[9];

  std::cout << einanderesFeld[9] << std::endl;

  return 0;
} // main( int nargs, char **args )

// Ende von tarr.cpp
```

5. Was gibt das folgende Programm aus (bitte mit Begründung!)?

```
// Time-stamp: "09.01.05 22:01 ausgabe.cpp klaus@wachtler.de"

#include <iostream>
#include <ostream>

class A
{
public:

  // Konstruktor
  A()
    : ersterwert( 0 )
  {
    std::cerr << "ctor A()" << std::endl;
  }

  A( int ersterwert )
    : ersterwert( ersterwert )
  {
    std::cerr << "ctor A( " << ersterwert << " )"
              << std::endl;
  }

  // copy-Konstruktor
  A( const A &rechteSeite )
    : ersterwert( rechteSeite.ersterwert )
  {
    std::cerr << "copy-ctor A( const A &rechteSeite )"
              << " rechteSeite.ersterwert = "
              << rechteSeite.ersterwert
              << std::endl;
  }

  // Destruktor
  virtual ~A()
  {
    std::cerr << "dtor ~A(), " << ersterwert << std::endl;
  }

  // Zuweisung A = A
  // Rückgabewert ist eine REFERENZ auf die linke Seite
  // (nicht const), und damit wieder als lvalue verwendbar.
  A &operator=( const A &rechteSeite )
  {
    std::cerr << "Zuweisung A=A, ls=" << ersterwert
```

```
              << ", rs=" << rechteSeite.ersterwert
              << std::endl;

    ersterwert = rechteSeite.ersterwert;
    return *this;
  }

  // Ausgabe auf einen ostream
  virtual void streamout( std::ostream & s ) const
  {
    s << "A=" << ersterwert;
  }

  // Kopie von sich selbst allokieren
  virtual A *clone() const
  {
    std::cout << "A::clone()" << std::endl;
    return new A(*this);
  }

protected:

  // Membervariable (in den abgeleiteten Klassen kommen noch welche
  // dazu):
  int ersterwert;

}; // class A...


class B: public A
{
public:

  // Konstruktor
  B()
    : A(),
      zweiterwert( 0 )
  {
    std::cerr << "ctor B()" << std::endl;
  }

  B( int ersterwert, int zweiterwert )
    : A( ersterwert ),
      zweiterwert( zweiterwert )
  {
    std::cerr << "ctor B( "
              << ersterwert
```

```
            << ", "
            << zweiterwert
            << " )"
            << std::endl;
}

// copy-Konstruktor
B( const B &rechteSeite )
  : A( rechteSeite.ersterwert ),
    zweiterwert( rechteSeite.zweiterwert )
{
  std::cerr << "copy-ctor B( const B &rechteSeite )"
            << "\n  rechteSeite.ersterwert = "
            << rechteSeite.ersterwert
            << ",\n  rechteSeite.zweiterwert = "
            << rechteSeite.zweiterwert
            << std::endl;
}

// Destruktor
virtual ~B()
{
  std::cerr << "dtor ~B()"
            << ", ersterwert=" << ersterwert
            << ", zweiterwert=" << zweiterwert
            << std::endl;
}

// Zuweisung B = B
// Rückgabewert ist eine REFERENZ auf die linke Seite
// (nicht const), und damit wieder als lvalue verwendbar.
B & operator=( const B &rechteSeite )
{
  std::cerr << "Zuweisung B=B"
            << ",\n  ls.ersterwert=" << ersterwert
            << ",\n  ls.zweiterwert=" << zweiterwert
            << ",\n  rs.ersterwert=" << rechteSeite.ersterwert
            << ",\n  rs.zweiterwert=" << rechteSeite.zweiterwert
            << std::endl;

  this->A::operator=( rechteSeite );
  zweiterwert = rechteSeite.zweiterwert;

  return *this;
}

// Ausgabe auf einen ostream
```

```
  virtual void streamout( std::ostream & s ) const
  {
    s << "B=";
    A::streamout( s );
    s << ", "  << zweiterwert;
  }

  // Kopie von sich selbst allokieren
  virtual B *clone() const
  {
    std::cout << "B::clone()" << std::endl;
    return new B(*this);
  }

protected:

  // Membervariable
  int zweiterwert;

}; // class B...

inline std::ostream &operator<<( std::ostream &s, const A &obj )
{
  obj.streamout( s );
  return s;
}

int main( int nargs, char **args )
{
  std::cout << "es geht los..." << std::endl;
  B einb( 12, 20 );
  std::cout << "new:" << std::endl;
  B *p_b = new B( 1, 2 );
  std::cout << "Ausgabe:" << std::endl;
  std::cout << "*p_b = " << *p_b << std::endl;
  B nocheinb = einb;
  std::cout << "nocheinb = " << nocheinb << std::endl;
  nocheinb = *p_b;
  std::cout << "nocheinb = " << nocheinb << std::endl;
  std::cout << "clone:" << std::endl;
  A *p_keineahnung = &einb;
  A *p_keineahnung2 = p_keineahnung->clone();
  std::cout << "*p_keineahnung2 = " << *p_keineahnung2 << std::endl;
  std::cout << "fertig!" << std::endl;
} // main( int nargs, char **args )
```

. . . beabsichtigte Leerseite. . .

6. Was gibt das folgende Programm aus (bitte mit Begründung!)?

```
#include <stdio.h>

char *c[]  =
  {
    "he dast ga",
    "lllt so doof",
    "C i",
    "dar nich"
  };

char **cp[] =
  {
    c+3,
    c+2,
    c+1,
    c
  };

char ***cpp  =   cp;

int main( int Anzarg, char **Arg )
{
  printf( "%s", **++cpp );
  printf( "%s", *--*++cpp+5 );
  printf( "%s", cpp[-2][0]+2 );
  printf( "%s\n", *(cpp[1]+1)+3 );
  return 0;
}
```

....

....

....

....

....

....

....

....

....

....

....

....

Ende der Aufgaben, endlich!