Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Studiengang Sensorik/Sensorsystemtechnik

Note / normierte Punkte

Klausur in Programmieren

Sommer 2012, 18. Juli 2012

Dauer: 1,5h

Hilfsmittel: Keine (Wörterbücher sind auf Nachfrage erlaubt)

Name:
MatrikelNr.:

Aufgabe	1	2	3	4	5	6	Summe
Punkte max	8	15	20	19	23	15	100
Punkte							

Alle Fragen beziehen sich auf den Stoff der Vorlesung. Somit sind sie z.B. bezogen auf die Programmiersprache C++. Auch sonst gelten die Konventionen wie in unserer Vorlesung.

1. Aufgabe: Grundlagen

a) Geben Sie an, welcher der C/C++ Ausdrücke eine Variablendefinition (F) und welche eine Variablendeklaration (D) ist. (4 P):

1) #define A 5 ..... ___----_____
2) double d = 9.81; ..... ___ F _____
3) int d; ..... ___ D _____
4) cout << “Definition:“; ..... ___---- _____

b) Welche Werte enthalten die nachfolgenden Variablen: (4 P)

1) int i = 25.7 * 2.0; i == __51_____

2) double d = 3.5 * 3; d == __10.5_____

3) int i = 7;
   double d = ((i++ * ((i +1) % 3)) – 1) / 2.0; i == __8___, d == __6.5__

2. Aufgabe: Grundlagen

a) Schreiben Sie ein Hauptprogramm, das die potentielle Energie eines Körpers berechnet. Lesen Sie dazu von der Konsole die Masse m in [kg] sowie die Höhe der Position des Körpers h in [m] ein. Die Erdbeschleunigung definieren Sie als globale Konstante mit dem Wert 9.81 in [m / s^2]. Berechnen Sie nun mit der Formel E = m * g * h die potentielle Energie des Körpers, speichern das Ergebnis in einer passenden Variablen und geben anschließend diesen Wert auf die Konsole aus. (9 P)

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

const double gcdErdbeschleunigung = 9.81;

int main()
{
  double dMasse;
  double dHoehe;

  cout << "Masse: "; cin >> dMasse;
  cout << "Hoehe: "; cin >> dHoehe;

  double dEnergie = dMasse * dHoehe * gcdErdbeschleunigung;

  cout << "potentielle Energie = " << dEnergie << endl;

  return 0;
}
```

b) Welche Schleifen gibt es in C/C++? Geben Sie kleine Beispiele: (6 P)

```
for(int li=0; li < 20; ++li)
{
}

int li=0;
while(li < 20)
{
    ++li;
}
```

```
int li=0;
do
{
   ++li;
}
while(li < 20);
```

3. Aufgabe: Funktionen

a) Erklären Sie aus welchen Teilen eine Funktion besteht (strukturiert vorgehen!) (6 P):
- Funktionskopf, Funktionsrumpf
- Funktionskopf besteht aus
  - Datentyp des Rückgabewertes
  - Funktionsname
  - Parameterliste
    - Parameterliste besteht aus keinem, einem, beliebig vielen Parametern, die durch Komma getrennt werden
    - Parameter besteht aus Datentyp und Parametername

b) Geben Sie ein Beispiel für eine Funktion, bei der genau ein Parameter mittels „call by value“ übergeben wird und ein double Ergebniswert mittels return zurückgegeben wird (nur ein Programmfragment, keine zusätzliche main-Routine, keine Ein-/Ausgabe) (4 P):

```
double beispiel(double dInValue)
{
   return dInValue * 12.1;
}
```

c) Geben Sie ein Beispiel, bei der eine Zeichenkette (Array mit Basistyp char) mittels „call by reference“ übergeben wird. Es soll kein Rückgabewert zurück gegeben werden (nur ein Programmfragment, keine zusätzliche main-Routine, keine Ein-/Ausgabe) (4 P).

```
void uppercase(char *  poInString)
{
// in Großbuchstaben ändern
}
```

d) Schreiben Sie eine vollständige Funktion, bei der die Summe von einem integer-Startwert bis zu einem integer-Endwert berechnet wird. Start- und End-Wert werden als Parameter übergeben, Ergebnis als Rückgabewert der Funktion (nur ein Programmfragment, keine zusätzliche main-Routine, keine Ein-/Ausgabe) (6 P).

```
int sum(int iInStartValue, int iInEndValue)
{
   int iSum = 0;
   for(int li=iInStartValue; li<=iInEndValue; ++li)
   {
      iSum += li;
   }
   return iSum;
}
```

4. Aufgabe: Array/Feld, Indizierung

a) Erklären Sie den Unterschied zwischen einem statischen und einem dynamischen Feld und was dabei generell zu beachten ist. Geben Sie jeweils ein einfaches Beispiel mit der Deklaration eines Feldes mit 10 int-Feldelementen: (10 P)

Statisches Feld:

Anzahl Feldelemente steht bei der Deklaration des Feldes zum Zeitpunkt des Programmierens fest und wird deshalb nur als Konstante akzeptiert. Speicher wird automatisch bereit gestellt und am Ende eines Anweisungsblocks wieder frei gegeben.

```
int aiFeld[10];
```

Dynamisches Feld:

Anzahl Feldelemente kann zur Laufzeit bestimmt werden. Speicher muss mittels new angefordert und durch delete freigegeben werden.

```
int*  aiFeld = new int[10];

…

delete [ ] aiFeld;
```

b) Gegeben ist folgender Funktionskopf: int maximum(int* aiInDaten, unsigned int uiInAnzahl) Schreiben Sie den dazu passenden Funktionsrumpf, der als Rückgabewert den größten Wert des Feldes zurück gibt. Der Parameter uiInAnzahl ist dabei immer größer 0 und enthält die Anzahl der Feldelemente. Sie können davon ausgehen, dass das Feld aiInDaten auch genau so viele Feldelemente enthält, wie uiInAnzahl angibt. (9 P)

```
int maximum(int* aiInDaten, unsigned int uiInAnzahl)
{
  int iMax = aiInDaten[0];
  for(unsigned int li = 1; li < uiInAnzahl; ++li)
  {
    if(iMax < aiInDaten[li])
    {
      iMax = aiInDaten[li];
    }
  }
  return iMax;
}
```

5. Aufgabe: Zeichenketten

a) Schreiben Sie eine Funktion strlength, die die Länge einer mit 0 terminierten Zeichenkette bestimmt. Der Funktionswert soll die Länge zurückgeben (kein Hauptprogramm, keine Ein- oder Ausgabe!). (8 P)

```
int strlength(char* acInString)
{
  int iLen=0;
  while(acInString[iLen] != 0)
  {
    iLen++;
  }
  return iLen;
}
```

b) Schreiben Sie eine Funktion charCount, die bestimmt, wie oft ein als Parameter übergebenes Zeichen in einer Zeichenkette vorkommt. Kommt das Zeichen gar nicht vor, soll 0 zurückgegeben werden (kein Hauptprogramm, keine Ein- oder Ausgabe!). (15 P)

```
int charCount(char cInChar, char* acInString)
{
  int iCount=0;
  while(*acInString != 0)
  {
    if(cInChar == *acInString++)
    {
      iCount++;
    }
  }
  return iCount;
}

oder:

int charCount(char cInChar, char* acInString)
{
  int iCount=0;
  int iLen = 0;
  while(acInString[iLen] != 0)
  {
    if(cInChar == acInString[iLen])
    {
      iCount++;
    }
    iLen++;
  }
  return iCount;
}
```

6. Aufgabe: Algorithmus

Was berechnet die nachfolgende Funktionen unknown1? Und wozu dienen die Funktionen unknown2 und unknown3?

Bitte beschreiben Sie die Funktionsweise möglichst abstrakt – Romane geben Abzug! (15 P)

```
int unknown1(char* acInChars1, char* acInChars2)
{
  int iIndex = 0;
  while(acInChars1[iIndex] == acInChars2[iIndex] && acInChars1[iIndex] != 0)
  {
    iIndex++;
  }
  if(acInChars1[iIndex] == acInChars2[iIndex])
  {
    return 0;
  }
  else
  if(acInChars1[iIndex] < acInChars2[iIndex])
  {
    return -1;
  }
  return 1;
}

void unknown2(char* acInChars1, char* acInChars2, int iInValue)
{
  cout << "unknown1('" << acInChars1 << "', '" << acInChars2 << "'): ";
  if(unknown1(acInChars1, acInChars2) == iInValue)
  {
    cout << "NOT " << endl;
  }
  cout << " FAILED!" << endl;
}

void unknown3()
{
  unknown2("Alfons", "Alfred", -1);
  unknown2("Alfons", "Alfons", 0);
  unknown2("Alfred", "Alfons", 1);
}
```

```
unknown1: Vergleich zweier Zeichenketten
          -1, wenn Zeichenkette1 alphabetisch vor Zeichenkette2
           0, wenn Zeichenkette1 gleich Zeichenkette2
           1, wenn Zeichenkette1 alphabetisch nach Zeichenkette2

unknown2: Routine, um Testfälle von unknown1 zu prüfen
          testCase_strcmp (Unit-Tests)

unknown3: Test-Routine (Unit-Tests) mit allen Testfällen, die
          geprüft werden sollen.
```