```
1  //============================================================================
2  //        * Letsch Informatik *       www.LetsInfo.ch       CH-8636 Wald
3  //          Beratung, Ausbildung und Realisation in Software-Engineering
4  //============================================================================
5  // Project   : Master of Advanced Studies in Software-Engineering MAS-SE 2019
6  // Modul     : C++
7  // Title     : Übung Zeiger/Referenzen und "Klasse Person": Lösung
8  // Author    : Thomas Letsch
9  // Tab-Width : 2
10 /*///=========================================================================
11 * Description: Detaildesign mit Zeigern und Referenzen.
12 *              Klasse implementieren und erweitern aufgrund applikatorischer
13 *              Anforderungen.
14 * $Revision  : 1.28 $  $Date: 2019/10/31 10:19:23 $
15 /*///=========================================================================
16 //       1         2         3         4         5         6         7         8
17 //345678901234567890123456789012345678901234567890123456789012345678901234567890
18 //============================================================================
19
20 #include <iostream>
21 #include <string>
22 #include <chrono>
23 #include <iomanip>
24 #include "Person.h"
25
26 using namespace std;
27
28 void aufgabe1();
29 void aufg1_funcPtr(string** str);
30 void aufg1_funcRef(string*& str);
31 void aufgabe2();
32 void aufgabe3();
33 void aufgabe4();
34
35 int main(int argc, char* argv[]) {
36   aufgabe1();
37   aufgabe2();
38   aufgabe3();
39   aufgabe4();
40   return 0;
41 }
42
43
44 void aufgabe1() {
45   // Detaildesign mit Zeigern und Referenzen:
46   cout << endl << "Aufgabe 1:" << endl;
47
48   string* strPtr = new string("NOT DONE YET");
49   aufg1_funcPtr(&strPtr);
50   cout << "str mit Zeiger   = " << *strPtr << endl;
51   strPtr = new string("NOT DONE YET");
52   aufg1_funcRef(strPtr);
53   cout << "str mit Referenz = " << *strPtr << endl;
54
55 }
56
57
58 void aufg1_funcPtr(string** str) {
59   string* tmpStrPtr = new string("funcPtr");
60   *str = tmpStrPtr;
61 }
62
63
64 void aufg1_funcRef(string*& str) {
65   str = new string("funcRef");
66 }
67
68
69
```



```
69 void aufgabe2() {
70   // Klasse, Strings, char-Array, Arrays von Objekten:
71   cout << endl << "Aufgabe 2:" << endl;
72
73   const int MAX_ARR = 5;
74
75   Person persArr[MAX_ARR];
76
77   cout << endl << "Personen-Array:" << endl;
78   for (int i = 0; i < MAX_ARR; i++) {
79     cout << persArr[i].getNr() << ": " << persArr[i].getName() << endl;
80   }
81   persArr[0].setName("Miller");
82   persArr[0].setNr(1);
83   persArr[1].setName("Bond");
84   persArr[1].setNr(007);
85   cout << endl << "Personen-Array:" << endl;
86   for (int i = 0; i < MAX_ARR; i++) {
87     cout << persArr[i].getNr() << ": " << persArr[i].getName() << endl;
88   }
89
90   // Ausgabe des Personen-Arrays wie oben mit for-Schleifen, jetzt aber mit
91   // Funktion 'printPersArr()':
92   cout << endl << "Personen-Array mit \'printPersArr()\':" << endl;
93   printPersArr(persArr, MAX_ARR);
94 }
95
96
97 void aufgabe3() {
98   // Untersuchung wieviel Zeit Objekt-Allozierungen auf Stack und Heap benoetigen:
99   cout << endl << "Aufgabe 3: Stack vs. Heap:" << endl;
100
101  chrono::time_point<std::chrono::high_resolution_clock> start, end;
102  const int MAX_LOOP = 10000000;
103
104  // Allozierung auf dem Stack:
105  start = chrono::high_resolution_clock::now();
106  for (int i = 0; i < MAX_LOOP; i++) {
107    Person pers(i, "John");
108  }
109  end = chrono::high_resolution_clock::now();
110  int64_t timeStack = chrono::duration_cast<chrono::milliseconds>(end-start).count();
111  cout << std::left << setw(12) << "Stack" << "= " << timeStack << " ms" << endl;
112
113  // Allozierung auf dem Heap:
114  start = chrono::high_resolution_clock::now();
115  for (int i = 0; i < MAX_LOOP; i++) {
116    Person* pers = new Person(i, "John");
117    delete pers;
118  }
119  end = chrono::high_resolution_clock::now();
120  int64_t timeHeap = chrono::duration_cast<chrono::milliseconds>(end-start).count();
121  cout << std::left << setw(12) << "Heap: new()" << "= " << timeHeap << " ms" << endl;
122 }
123
124
125 void aufgabe4() {
126   // Links-Shift-Operator (<<) und 'const':
127   cout << endl << "Aufgabe 4:" << endl;
128
129   Person tom(1, "Tom");
130   Person john = "John Smith";
131   cout << tom << endl;
132   cout << john << endl;
133
134   const Person bond(4711, "James Bond");
135   cout << bond << endl;
136   cout << bond.getNr();
137 }
138
139
```

31.10.2019 10:19:27 **PersonTest.cpp** Page 3/3

```
/* Session-Log:

$ make clean all
rm -f Person.o PersonTest.o appl.exe
g++ -g -std=c++11 -c Person.cpp
g++ -g -std=c++11 -c PersonTest.cpp
g++ -g -std=c++11 -o appl.exe Person.o PersonTest.o

$ ./appl.exe

Aufgabe 1:
str mit Zeiger   = funcPtr
str mit Referenz = funcRef

Aufgabe 2:

Personen-Array:
-1:
-1:
-1:
-1:
-1:

Personen-Array:
1: Miller
7: Bond
-1:
-1:
-1:

Personen-Array mit 'printPersArr()':
1: Miller
7: Bond
-1:
-1:
-1:

Aufgabe 3: Stack vs. Heap:
Stack       = 126 ms
Heap: new() = 523 ms

Aufgabe 4:
Name: Tom                   Nr: 1
Name: John Smith            Nr: -1
Name: James Bond            Nr: 4711
4711

$

*/
```

31.10.2019 10:10:53 **Makefile** Page 1/1

```

# Makefile für Übung "Klasse Person"                              31.10.2019

BIN    = appl.exe
OBJS   = Person.o PersonTest.o

CC     = g++
CFLAGS = -g -std=c++11

all: $(BIN)

$(BIN): $(OBJS)
  $(CC) $(CFLAGS) -o $(BIN) $(OBJS)

clean:
  rm -f $(OBJS) $(BIN)

%.o: %.cpp %.h
  $(CC) $(CFLAGS) -c $<

%.o: %.cpp
  $(CC) $(CFLAGS) -c $<

```

31.10.2019 10:16:02 **Person.h** Page 1/1

```
#ifndef PERSON_H
#define PERSON_H 1

#include <iostream>

using std::ostream;


// Klassen-Definition:

// Person:
// Eine Person hat einen Namen (20 Char's) und eine Nummer.

const int NAME_LEN = 20;

class Person {

  enum {NAME_LEN = 20};  // andere Variante:
                         // Gültigkeitsbereich auf Klasse beschraenkt.

  friend ostream& operator<<(ostream& pOS, const Person& pPerson);

  public:
                 Person(const char* pName = nullptr);
                 Person(int pNr, const char* pName);
    int          getNr() const;
    const char*  getName() const;
    void         setNr(int pNr);
    void         setName(const char* pName);

  private:
    int   mNr;
    char  mName[NAME_LEN+1];

};


// Funktions-Prototypen:

// Ausgabe eines Arrays von Personen-Objekten auf die Konsole.
//   pPers: Pointer auf erstes Personen-Objekt im Array.
//   pLen:  Laenge des Personen-Arrays.
void printPersArr(Person* pPers, int pLen);


#endif /*PERSON_H*/
```

31.10.2019 10:15:56 **Person.cpp** Page 1/1

```

#include "Person.h"
#include <cstring>
#include <iomanip>

using std::cout;
using std::endl;
using std::setw;
using std::left;


// Methoden-Implementationen:


Person::Person(const char* pName) : mNr(-1) {
  if (pName == nullptr) {
    mName[0] = '\0';  // Null-Byte als erstes Byte im Array.
  }
  else {
    setName(pName);
  }
}


Person::Person(int pNr, const char* pName) : mNr(pNr) {
  setName(pName);
}


int Person::getNr() const {
  return mNr;
}


const char* Person::getName() const {
  return mName;
}


void Person::setNr(int pNr) {
  mNr = pNr;
}


void Person::setName(const char* pName) {
  strcpy(mName, pName);
}


// Funktions-Implementationen:

void printPersArr(Person* pPers, int pLen) {
  for (int i = 0; i < pLen; i++) {
    cout << pPers->getNr() << ": " << pPers->getName() << endl;
    pPers++;  // pPers soll auf 'naechste' Person zeigen (-> Zeiger-Arithmetik)
  }
}


// Operator als globale Funktion:

ostream& operator<<(ostream& pOutputStream, const Person& pPerson) {
  pOutputStream <<  "Name: " << setw(Person::NAME_LEN) << left << pPerson.mName
                << "  Nr: " << pPerson.mNr;
  return pOutputStream;
}
```