```
  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 2010
  6 // Modul     : C++
  7 // Title     : Übung "Klasse Person": Lösung
  8 // Author    : Thomas Letsch
  9 // Tab-Width : 2
 10 /*///=========================================================================
 11 * Description: Klasse implementieren und erweitern aufgrund applikatorischer
 12 *              Anforderungen.
 13 * History    : 11.01.04: Initial Version.
 14 *              29.10.05: Anpassung wegen Eclipse/CDT-Bug
 15 *                        (siehe Bugzilla Bug #102043)
 16 *              01.10.07: Anpassung wegen Eclipse/CDT-Bug wieder entfernt.
 17 *              02.11.08: Merge für MAS.
 18 *              19.09.10: inline -> Optimierung
 19 * Version    : $Revision: 1.13 $  $Date: 2010/09/19 15:36:33 $
 20 /*///=========================================================================
 21 //       1         2         3         4         5         6         7         8
 22 //345678901234567890123456789012345678901234567890123456789012345678901234567890
 23 //============================================================================
 24
 25 #include <iostream>
 26 #include <cstring>
 27 #include "Person.h"
 28 #include "StatisticStopWatch.h"
 29
 30 using namespace std;
 31
 32 void aufgabe1();
 33 void aufgabe2();
 34 void aufgabe4();
 35
 36 int main(int argc, char* argv[]) {
 37
 38   if ((argc == 2) && (strcmp(argv[1], "OnlyStackVsHeap") == 0)) {
 39     aufgabe2();
 40   }
 41   else {
 42     aufgabe1();
 43     aufgabe2();
 44     aufgabe4();
 45   }
 46   return 0;
 47
 48 }
 49
 50
```



```
 51 void aufgabe1() {
 52   // Klasse, Strings, char-Array, Arrays von Objekten:
 53   cout << endl << "Aufgabe 1:" << endl;
 54
 55   const int MAX_ARR = 5;
 56
 57   Person persArr[MAX_ARR];
 58   cout << endl << "Personen-Array:" << endl;
 59   for (int i = 0; i < MAX_ARR; i++) {
 60     cout << persArr[i].getNr() << ": " << persArr[i].getName() << endl;
 61   }
 62   persArr[0].setName("Miller");
 63   persArr[0].setNr(1);
 64   persArr[1].setName("Bond");
 65   persArr[1].setNr(007);
 66   cout << endl << "Personen-Array:" << endl;
 67   for (int i = 0; i < MAX_ARR; i++) {
 68     cout << persArr[i].getNr() << ": " << persArr[i].getName() << endl;
 69   }
 70
 71   // Ausgabe des Personen-Arrays wie oben mit for-Schleifen, jetzt aber mit
 72   // Funktion 'printPersArr()':
 73   cout << endl << "Personen-Array mit \'printPersArr()\':" << endl;
 74   printPersArr(persArr, MAX_ARR);
 75 }
 76
 77
 78 void aufgabe2() {
 79   // Untersuchung wieviel Zeit Objekt-Allozierungen auf Stack und Heap benoetigen:
 80   cout << endl << "Aufgabe 2: Stack vs. Heap:" << endl;
 81
 82   StatisticStopWatch stopWatch;
 83   const int MAX_LOOP = 10000000;
 84
 85   // Allozierung auf dem Stack:
 86   stopWatch.reset();
 87   stopWatch.start();
 88   for (int i = 0; i < MAX_LOOP; i++) {
 89     Person pers(i, "John");
 90   }
 91   stopWatch.stop();
 92   stopWatch.printTime("Stack");
 93
 94   // Allozierung auf dem Heap:
 95   stopWatch.reset();
 96   stopWatch.start();
 97   for (int i = 0; i < MAX_LOOP; i++) {
 98     Person* pers = new Person(i, "John");
 99   }
100   stopWatch.stop();
101   stopWatch.printTime("Heap: new()");
102 }
103
104
105 void aufgabe4() {
106   // Links-Shift-Operator (<<) und 'const':
107   cout << endl << "Aufgabe 4:" << endl;
108
109   Person tom(1, "Tom");
110   Person john = "John Smith";
111   cout << tom << john;
112
113   const Person bond(4711, "James Bond");
114   cout << bond;
115   cout << bond.getNr();
116 }
117
118
```

```
/* Session-Log:

$ make clean all
rm -f Person.o PersonTest.o StatisticStopWatch.o appl.exe
g++ -g -O0 -c Person.cpp
g++ -g -O0 -c PersonTest.cpp
g++ -g -O0 -c StatisticStopWatch.cpp
g++ -g -O0 -o appl.exe Person.o PersonTest.o StatisticStopWatch.o

$ ./appl.exe

Aufgabe 1:

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 2: Stack vs. Heap:
Stack                =        250 ms
Heap: new()          =       2040 ms

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


# Aufgabe 3:

$ for i in O0 O1 O2 O3
> do
>   echo ""
>   echo "Optimization $i:"
>   g++ -$i -o appl.exe Person.cpp PersonTest.cpp StatisticStopWatch.cpp
>   ./appl.exe OnlyStackVsHeap
> done

Optimization-Level: O0 :

Aufgabe 2: Stack vs. Heap:
Stack                =        256 ms
Heap: new()          =       2031 ms

Optimization-Level: O1 :

Aufgabe 2: Stack vs. Heap:
Stack                =        220 ms
Heap: new()          =       2015 ms

Optimization-Level: O2 :

Aufgabe 2: Stack vs. Heap:
Stack                =        188 ms
Heap: new()          =       1937 ms

Optimization-Level: O3 :

Aufgabe 2: Stack vs. Heap:
Stack                =        150 ms
Heap: new()          =       1920 ms

*/
```

```
 1 #ifndef PERSON_H
 2 #define PERSON_H 1
 3
 4 #include <cstring>
 5 #include <iostream>
 6
 7 using std::ostream;
 8
 9
10 // Klassen-Definition:
11
12 // Person:
13 // Eine Person hat einen Namen (20 Char's) und eine Nummer.
14
15 const int NAME_LEN = 20;
16
17 class Person {
18
19   enum {NAME_LEN = 20};  // andere Variante:
20                          // Gültigkeitsbereich auf Klasse beschraenkt.
21
22
23   friend ostream& operator<<(ostream& pOS, const Person& pPerson);
24
25   public:
26                   Person(const char* pName = 0);
27                   Person(int pNr, const char* pName);
28     int           getNr() const;
29     const char*   getName() const;
30     void          setNr(int pNr);
31     void          setName(const char* pName);
32
33   private:
34     int   mNr;
35     char  mName[NAME_LEN+1];
36
37 };
38
39
40 // Funktions-Prototypen:
41
42 // Ausgabe eines Arrays von Personen-Objekten auf die Konsole.
43 //   pPers: Pointer auf erstes Personen-Objekt im Array.
44 //   pLen:  Laenge des Personen-Arrays.
45 void printPersArr(Person* pPers, int pLen);
46
47
48 #endif /*PERSON_H*/
```



```
 1
 2 #include "Person.h"
 3 #include <iomanip>
 4
 5 using std::ostream;
 6 using std::cout;
 7 using std::endl;
 8 using std::setw;
 9 using std::left;
10
11 // Methoden-Implementationen:
12
13
14 Person::Person(const char* pName) : mNr(-1) {
15   if (pName == 0) {
16     mName[0] = '\0';  // Null-Byte als erstes Byte im Array.
17   }
18   else {
19     setName(pName);
20   }
21 }
22
23
24 Person::Person(int pNr, const char* pName) : mNr(pNr) {
25   setName(pName);
26 }
27
28
29 int Person::getNr() const {
30   return mNr;
31 }
32
33
34 const char* Person::getName() const {
35   return mName;
36 }
37
38
39 void Person::setNr(int pNr) {
40   mNr = pNr;
41 }
42
43
44 void Person::setName(const char* pName) {
45   strcpy(mName, pName);
46 }
47
48
49 // Funktions-Implementationen:
50
51 void printPersArr(Person* pPers, int pLen) {
52   for (int i = 0; i < pLen; i++) {
53     cout << pPers->getNr() << ": " << pPers->getName() << endl;
54     pPers++;  // pPers soll auf 'naechste' Person zeigen (-> Zeiger-Arithmetik)
55   }
56 }
57
58
59 // Operator als globale Funktion:
60
61 ostream& operator<<(ostream& pOutputStream, const Person& pPerson) {
62   pOutputStream <<  "Name: " << setw(Person::NAME_LEN) << left << pPerson.mName
63                 << "  Nr: " << pPerson.mNr << "\n";
64   return pOutputStream;
65 }
```