UFPE – CENTRO DE INFORMÁTICA
ESTRUTURAS DE DADOS ORIENTADAS A OBJETOS
ATIVIDADE PRÁTICA

(Gustavo Carvalho – ghpc@cin.ufpe.br)

Exercícios selecionados de: KIRCH-PRINZ, U., PRINZ, P.
A Complete Guide to Programming in C++.
1a Edição. Editora Jones & Bartlett Learning, 2001.

Exercise 1

A program needs a class to represent the date.

- Define the class `Date` for this purpose using three integral data members for day, month, and year. Additionally, declare the following methods:

  ```
  void init( int month, int day, int year);
  void init(void);
  void print(void);
  ```

  Store the definition of the class Date in a header file.

- Implement the methods for the class Date in a separate source file:
  - The method print() outputs the date to standard output using the format Month-Day-Year.
  - The method init() uses three parameters and copies the values passed to it to corresponding members. A range check is not required at this stage, but will be added later.
  - The method init() without parameters writes the current date to the corresponding members. Use the functions declared in `ctime`.

    ```
    time_t time(time_t *ptrSec);
    struct tm *localtime(const time_t *ptrSec);
    ```

    The structure tm and sample calls to this function are included in what follows. The type time_t is defined as long in ctime. The function time() returns the system time expressed as a number of seconds and writes this value to the variable referenced by ptrSec. This value can be passed to the function localtime() that converts the number of seconds to the local type tm date and returns a pointer to this structure.

- Test the class Date using an application program that once more is stored in a separate source file. To this end, define two objects for the class and display the current date. Use object assignments and—as an additional exercise—references and pointers to objects.

```
struct tm
{
      int tm_sec;
      int tm_min;
      int tm_hour;
      int tm_mday;
      int tm_mon;
      int tm_year;
      int tm_wday;
      int tm_yday;
      int tm_isdst;
};
```

Sample calls to functions time( ) and localtime( )

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

int main() {
      struct tm *ptr;
      time_t sec;
      . . .
      time(&sec);
      ptr = localtime(&sec);

      cout << "Today is the " << ptr->tm_yday + 1
              << ". day of the year " << ptr->tm_year
              << endl;
      . . .
}
```

Exercise 2

A warehouse management program needs a class to represent the articles in stock.

- Define a class called Article for this purpose using the data members and methods shown below. Store the class definition for Article in a separate header file. Declare the constructor with default arguments for each parameter to ensure that a default constructor exists for the class. Access methods for the data members are to be defined as inline. Negative prices must not exist. If a negative price is passed as an argument, the price must be stored as 0.0.

  Private members:
  - article number (`long`), article name (`string`), sales price (`double`).

Public members:

- `Article(long, const string&, double);`
- `~Article();`
- `void print();` // Formatted output
- `set`- and `get`-methods for any data member

- Implement the constructor, the destructor, and the method print() in a separate source file. Also define a global variable for the number of Article type objects.

- The constructor must use the arguments passed to it to initialize the data members, additionally increment the global counter, and issue the message shown below.

```
An object of type Article . . . is created.
This is the . . . Article.
```

- The destructor also issues a message and decrements the global counter. The method print() displays a formatted object on screen. After outputting an article, the program waits for the return key to be pressed.

```
The object of type Article . . . is destroyed.
There are still . . . articles.
```

- The application program (again use a separate source file) tests the Article class. Define four objects belonging to the Article class type:

    a. A global object and a local object in the main function.
    b. Two local objects in a function test() that is called twice by main(). One object needs a static definition. The function test() displays these objects and outputs a message when it is terminated.

  Use articles of your own choice to initialize the objects. Additionally, call the access methods to modify individual data members and display the objects on screen.

- Test your program. Note the order in which constructors and destructors are called.

- Supplementary question: Suppose you modify the program by declaring a function called test() with a parameter of type Article and calling the function with an article type object. The counter for the number of objects is negative after running the program. Why?

Exercise 3

In the previous exercise you defined a simple class called Article.This involved using a global counter to log object creation and destruction. Improve and extend the Article class as follows:

- Use a static data member instead of a global variable to count the current number of objects.
- Declare a static access method called `getCount()` for the Article class. The method returns the current number of objects.
- Define a copy constructor that also increments the object counter by 1 and issues a message.This ensures that the counter will always be accurate. Tip: Use member initializers.

  Note: The copy constructor creates a copy of an existing object.The parameter is thus a read-only reference to the object that needs to be copied. The copy constructor in the Article class is thus declared as follows: `Article(const Article&);` The default copy constructor simply transfers the data members to the new object.

- Test the new version of the class. To do so, call the function test() by passing an article type object to the function.

Exercise 4

To practice the use of arrays, write a program that outputs all prime numbers less than 1000.The program should also count the number of prime numbers less than 1000. An integer >= 2 is a prime number if it is not divisible by any number except 1 and itself. Use the Sieve of Eratosthenes. To find primary numbers simply eliminate multiples of any prime numbers you have already found, i.e.: first eliminate any multiples of 2 (4, 6, 8, ...), then eliminate any multiples of 3 (6, 9, 12, ...), then eliminate any multiples of 5 (10, 15, 20, ..) // 4 has already been eliminated and so on.