Aufgabenzettel 3

Kianusch Vahid Yousefnia, Raphael Senghaas und Jan Maintok
Tutor: **Enes Witwit**

Aufgabe 1

```cpp
#include <iostream>
#include <string>
#include <cassert>

using namespace std;

string easter(int year){
  //calculating constants as in given algorithm
  int a = year%19;
  int b = year%4;
  int c = year%7;
  int k = year/100;
  int p = (8*k+13)/25;
  int q = k/4;
  int m = (15+k-p-q)%30;
  int d = (19*a+m)%30;
  int n = (4+k-q)%7;
  int e = (2*b+4*c+6*d+n)%7;
  int x = 22+d+e;
  int z;
  /*
    case differentiation for value z, that has to be
    50 if x=57,
    49 if x=56 and d=28 and a>10,
    equal to x otherwise
  */
  (x==57 || (x==56 && d==28 && a>10) ) ?
    (x==57 ?
      z = 50 :
      z = 49 ) :
    z = x;

  string date;
  /*
    case differentiation for the output value, that has to be
    "z. Maerz" if z<32
    "(z-31). April" otherwise
  */
  ( z<32 ?
    date = to_string(z) + ". Maerz" :
    date = to_string(z-31) + ". April");

  return "Ostern ist am " + date;
}


int main(){
  //testing functionality
  assert(easter(1583)=="Ostern ist am 10. April");
  assert(easter(1602)=="Ostern ist am 7. April");
  assert(easter(2016)=="Ostern ist am 27. Maerz");
  assert(easter(2718)=="Ostern ist am 7. April");
  assert(easter(2998)=="Ostern ist am 8. April");
```

```
  assert(easter(3141)=="Ostern ist am 20. April");
  assert(easter(6283)=="Ostern ist am 15. April");
  assert(easter(6626)=="Ostern ist am 2. April");
  assert(easter(8314)=="Ostern ist am 12. April");
  assert(easter(1000000)=="Ostern ist am 16. April");
  return 0;
}
```

Aufgabe 2

```
//Packages
#include <iostream>
#include <cstring>
#include <cassert>

// Declarations
int weekday2001(int day,int month,int year);
int floorDiv(int a, int b);
int abs(int a);
int sgn(int a);
int floorMod(int a, int b);
int weekday(int day, int month, int year);
int weekday1(int day, int month, int year);

  /*
  Code for the days of the week:
  0 for Monday
  1 for Tuesday
  etc.
  */

int main (){

  // Exercise a)

  // Some sanity checks
  assert(2 == weekday2001(7,6,2006));
  assert(5 == weekday2001(24,2,2001));
  // Wrong result expected for the following
  //(as commentary after I checked and proceeded with further
  // subtasks)
  //assert(5 == weekday2001(21,11,1964));
  //assert(4 == weekday2001(13,5,1492));


  // Exercise c)

  /* Durch die Implementierung von floorMod wird erreicht,
  dass der Rest immer nichtnegativ ist, auch bei negativen
  Divisionen. Die %-Funktion wird bei ungleichen Vorzeichen
  von Zaehler und Nenner dagegen negativ, liegt aber in der
  selben Restklasse wie das Resultat der floorMod-Funktion,
  d.h. result_% + nenner = result_floorMod
  */


  // Exercise d)

  assert(4 == weekday(3,2,2012)); // Friday
  assert(2 == weekday(29,2,2012)); // Wednesday
  assert(2 == weekday(28,11,2012)); // Wednesday
  assert(5 == weekday(5,1,2013)); // Saturday
  assert(3 == weekday(7,3,2013)); // Thursday
  assert(6 == weekday(24,12,2017)); // Sunday
  assert(5 == weekday(8,1,2000)); // Saturday
```

```
  assert(0 == weekday(28,2,2000)); // Monday
  assert(5 == weekday(11,11,2000)); // Saturday
  assert(0 == weekday(19,2,1900)); // Monday
  assert(2 == weekday(1,8,1900)); // Wednesday
  assert(0 == weekday(28,2,1600)); // Monday
  assert(2 == weekday(27,9,1600)); // Wednesday
  assert(2 == weekday(1,3,1600)); // Wednesday
  assert(5 == weekday(27,2,2100)); // Saturday
  assert(2 == weekday(31,3,2100));  // Wednesday

  /* remark : Die auf dem Zettel vorgeschlagene
  Homepage akzeptiert keine 29. Februare trotz Schaltjahr
  */


  // Exercise e)

  /*
  Die Schaltjahresregelung ist zyklisch in 400 Jahresschritten,
  deshalb kann von yearsPassed - 2001 ein Vielfaches von 400
  addiert werden. Waehlt man 5*400=2000, so ist yearsPassed
  fuer Jahre nach 1583 (Einfuehrung greg. Kalender) immer positiv
  und damit gilt truncating division = floor division. Auch
  die Folgegroessen daysPassed und weekday sind dann so beschaffen,
  dass mit den Standard-Tools das Richtige herauskommt.
  */

  assert(4 == weekday1(3,2,2012)); // Friday
  assert(2 == weekday1(29,2,2012)); // Wednesday
  assert(2 == weekday1(28,11,2012)); // Wednesday
  assert(5 == weekday1(5,1,2013)); // Saturday
  assert(3 == weekday1(7,3,2013)); // Thursday
  assert(6 == weekday1(24,12,2017)); // Sunday
  assert(5 == weekday1(8,1,2000)); // Saturday
  assert(0 == weekday1(28,2,2000)); // Monday
  assert(5 == weekday1(11,11,2000)); // Saturday
  assert(0 == weekday1(19,2,1900)); // Monday
  assert(2 == weekday1(1,8,1900)); // Wednesday
  assert(0 == weekday1(28,2,1600)); // Monday
  assert(2 == weekday1(27,9,1600)); // Wednesday
  assert(2 == weekday1(1,3,1600)); // Wednesday
  assert(5 == weekday1(27,2,2100)); // Saturday
  assert(2 == weekday1(31,3,2100));  // Wednesday









  return 0;

}









// functions of my own

//Function to determine weekday,
 // using wrong implementation using truncating division:
int weekday2001(int day, int month, int year){
  //// what day was January 1st of the given year?
  // => taking into account leap year rules
  int yearsPassed = year - 2001;
```

```
  int weekdayJanuary1 = (365*yearsPassed +
    yearsPassed/4 - yearsPassed/100 + yearsPassed/400) % 7;

  ////Determine if given year is leap year:
  //leap year if divisible by 4 but not by 100, or if divisible by 400
  bool LeapYear = ((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0);


  //// How many days have passed since January 1st?
  int daysPassed = 0;
  (month == 1) // January?
    ? daysPassed = day
    : (month == 2) // February?
      ? daysPassed = day + 31
      : (month > 2 && LeapYear) // something else and leap year?
        ? daysPassed = day + 60 + (153*month - 457)/5
        : daysPassed = day + 59 + (153*month - 457)/5;

  //// Calculation of weekday
    int weekday = (weekdayJanuary1 + daysPassed - 1) % 7;


  return weekday;
}

//Function to determine absolute value of integer
int abs(int a){
  int result = 0;
  (a >= 0)
    ? result = a
    : result = -a;
  return result;
}

// Function to determine sign of integer
int sgn(int a){
  int result = 0;
  (a > 0)
    ? result = 1
    : (a < 0)
      ? result = -1
      : result = 0;
  return result;
}

// Function to round up
int floorDiv(int a, int b){
  // Find out if result is negative (use truncating
  //division if not)
  int result = 0;
  (sgn(a) == sgn(b))
    ? result = a/b
    : result = (-abs(a) - abs(b))/abs(b); //subtract 1 from fraction
                        //and round up
  return result;
}

// Function to calculate the remainder taking into account floorDiv
int floorMod(int a, int b){
  int result = a - b*floorDiv(a,b);
  return result;
}


// Function to determine weekday correctly
int weekday(int day, int month, int year){
```

```
  //// what day was January 1st of the given year?
  // => taking into account leap year rules
  int yearsPassed = year - 2001;
  int weekdayJanuary1 = floorMod((365*yearsPassed +
    floorDiv(yearsPassed,4) - floorDiv(yearsPassed,100)
    + floorDiv(yearsPassed,400)),7);

  ////Determine if given year is leap year:
  //leap year if divisible by 4 but not by 100, or if divisible by 400
  bool LeapYear = ((floorMod(year,4) == 0)
          && (floorMod(year,100) != 0))
          || (floorMod(year,400) == 0);


  //// How many days have passed since January 1st?
  int daysPassed = 0;
  (month == 1) // January?
    ? daysPassed = day
    : (month == 2) // February?
      ? daysPassed = day + 31
      : (month > 2 && LeapYear) // something else and leap year?
        ? daysPassed = day + 60 + floorDiv((153*month - 457),5)
        : daysPassed = day + 59 + floorDiv((153*month - 457),5);

  //// Calculation of weekday
    int weekday = floorMod((weekdayJanuary1 + daysPassed - 1),7);


  return weekday;
}

// Function for weekday without floorDiv and floorMod for subtask e)
int weekday1(int day, int month, int year){
  //// what day was January 1st of the given year?
  // => taking into account leap year rules
  int yearsPassed = year - 1;
  int weekdayJanuary1 = (365*yearsPassed +
    yearsPassed/4 - yearsPassed/100 + yearsPassed/400) % 7;

  ////Determine if given year is leap year:
  //leap year if divisible by 4 but not by 100, or if divisible by 400
  bool LeapYear = ((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0);


  //// How many days have passed since January 1st?
  int daysPassed = 0;
  (month == 1) // January?
    ? daysPassed = day
    : (month == 2) // February?
      ? daysPassed = day + 31
      : (month > 2 && LeapYear) // something else and leap year?
        ? daysPassed = day + 60 + (153*month - 457)/5
        : daysPassed = day + 59 + (153*month - 457)/5;

  //// Calculation of weekday
    int weekday = (weekdayJanuary1 + daysPassed - 1) % 7;


  return weekday;
}
```

Aufgabe 4

```
#include <iostream>
#include <string>
```

```cpp
#include <cassert>
#include <cmath>

using namespace std;

//calculating sqare of a number
double sq(double x){
  return x*x;
}

//recursive method to calculate powers of x
double power(double x, int n){
  assert(n>0);
  double ans;

  (n==1) ?
    ans = x :
    ((n%2 == 0) ?
      /*
        if the result for n beeing equal would be calculated
        as power(x,n/2)*power(x,n/2), the power function would
        be executed twice, this is exactly what we are trying to avoid
        to reduce the running time of the calculation, by only calculating
        it once and using the result instead of the function itself
        to calculate the square.
      */
      ans = sq(power(x,n/2)):
      ans = x*power(x,n-1));
    return ans;
}

/*
  function that prints the relevant parameters to compare
  the two methods of calculating powers of x
  The comparison between our own function and std::pow was done by
  printing the results instead of using the assert() function, because
  of the differences described below. This would crash the program for
  certain inut values.
*/
void testing(double x, int n, int i){
  cout << "test no." << to_string(i) << ":" << endl;
  cout << "to calculate: " << to_string(x) << "^" << to_string(n) << endl;
  cout << "result of own function: " << to_string(power(x,n)) << endl;
  cout << "result of std::pow():   " << to_string(std::pow(x,n)) << endl;
  cout << "difference:             " << to_string(power(x,n)-std::pow(x,n)) <<
    endl;
  cout << endl;
}

int main(){

  testing(2,2,1);
  testing(10,10,4);
  testing(3.141592,10,2);
  testing(42,13,5);
  testing(42,42,6);
  testing(314,100,7);
  testing(2.71828182846,42,8);
  testing(3.141592,42,3);
  /*
    interestingly the results for big powers of big or non integer
    numbers are NOT equal. This is most likely due to the fact, that
    the std::pow() function works with floats instead of doubles
  */
  return 0;
}
```