
Aufgabenzettel 3

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

Aufgabe 1

```
1 #include <iostream>
2 #include <string>
3 #include <cassert>
4
5 using namespace std;
6
7 string easter(int year){
8     //calculating constants as in given algorithm
9     int a = year%19;
10    int b = year%4;
11    int c = year%7;
12    int k = year/100;
13    int p = (8*k+13)/25;
14    int q = k/4;
15    int m = (15+k-p-q)%30;
16    int d = (19*a+m)%30;
17    int n = (4+k-q)%7;
18    int e = (2*b+4*c+6*d+n)%7;
19    int x = 22+d+e;
20    int z;
21    /*
22     case differentiation for value z, that has to be
23     50 if x=57,
24     49 if x=56 and d=28 and a>10,
25     equal to x otherwise
26    */
27    (x==57 || (x==56 && d==28 && a>10) ) ?
28        (x==57 ?
29            z = 50 :
30            z = 49 ) :
31        z = x;
32
33    string date;
34    /*
35     case differentiation for the output value, that has to be
36     "z. Maerz" if z<32
37     "(z-31). April" otherwise
38    */
39    ( z<32 ?
40        date = to_string(z) + ". Maerz" :
41        date = to_string(z-31) + ". April");
42
43    return "Ostern ist am " + date;
44 }
45
46
47 int main(){
48     //testing functionality
49     assert(easter(1583)=="Ostern ist am 10. April");
50     assert(easter(1602)=="Ostern ist am 7. April");
51     assert(easter(2016)=="Ostern ist am 27. Maerz");
52     assert(easter(2718)=="Ostern ist am 7. April");
53     assert(easter(2998)=="Ostern ist am 8. April");
```

```

54 assert(easter(3141)=="Ostern ist am 20. April");
55 assert(easter(6283)=="Ostern ist am 15. April");
56 assert(easter(6626)=="Ostern ist am 2. April");
57 assert(easter(8314)=="Ostern ist am 12. April");
58 assert(easter(1000000)=="Ostern ist am 16. April");
59 return 0;
60 }

```

Aufgabe 2

```

1 //Packages
2 #include <iostream>
3 #include <cstring>
4 #include <cassert>
5
6 // Declarations
7 int weekday2001(int day,int month,int year);
8 int floorDiv(int a, int b);
9 int abs(int a);
10 int sgn(int a);
11 int floorMod(int a, int b);
12 int weekday(int day, int month, int year);
13 int weekday1(int day, int month, int year);
14
15 /*
16  Code for the days of the week:
17  0 for Monday
18  1 for Tuesday
19  etc.
20  */
21
22 int main (){
23
24     // Exercise a)
25
26     // Some sanity checks
27     assert(2 == weekday2001(7,6,2006));
28     assert(5 == weekday2001(24,2,2001));
29     // Wrong result expected for the following
30     //(as commentary after I checked and proceeded with further
31     // subtasks)
32     //assert(5 == weekday2001(21,11,1964));
33     //assert(4 == weekday2001(13,5,1492));
34
35
36     // Exercise c)
37
38     /* Durch die Implementierung von floorMod wird erreicht,
39     dass der Rest immer nichtnegativ ist, auch bei negativen
40     Divisionen. Die %-Funktion wird bei ungleichen Vorzeichen
41     von Zaehler und Nenner dagegen negativ, liegt aber in der
42     selben Restklasse wie das Resultat der floorMod-Funktion,
43     d.h. result_% + nenner = result_floorMod
44     */
45
46
47     // Exercise d)
48
49     assert(4 == weekday(3,2,2012)); // Friday
50     assert(2 == weekday(29,2,2012)); // Wednesday
51     assert(2 == weekday(28,11,2012)); // Wednesday
52     assert(5 == weekday(5,1,2013)); // Saturday
53     assert(3 == weekday(7,3,2013)); // Thursday
54     assert(6 == weekday(24,12,2017)); // Sunday
55     assert(5 == weekday(8,1,2000)); // Saturday

```

```

56  assert(0 == weekday(28,2,2000)); // Monday
57  assert(5 == weekday(11,11,2000)); // Saturday
58  assert(0 == weekday(19,2,1900)); // Monday
59  assert(2 == weekday(1,8,1900)); // Wednesday
60  assert(0 == weekday(28,2,1600)); // Monday
61  assert(2 == weekday(27,9,1600)); // Wednesday
62  assert(2 == weekday(1,3,1600)); // Wednesday
63  assert(5 == weekday(27,2,2100)); // Saturday
64  assert(2 == weekday(31,3,2100)); // Wednesday
65
66  /* remark : Die auf dem Zettel vorgeschlagene
67  Homepage akzeptiert keine 29. Februare trotz Schaltjahr
68  */
69
70
71  // Exercise e)
72
73  /*
74  Die Schaltjahresregelung ist zyklisch in 400 Jahresschritten,
75  deshalb kann von yearsPassed - 2001 ein Vielfaches von 400
76  addiert werden. Waehlt man 5*400=2000, so ist yearsPassed
77  fuer Jahre nach 1583 (Einfuehrung greg. Kalender) immer positiv
78  und damit gilt truncating division = floor division. Auch
79  die Folgegroessen daysPassed und weekday sind dann so beschaffen,
80  dass mit den Standard-Tools das Richtige herauskommt.
81  */
82
83  assert(4 == weekday1(3,2,2012)); // Friday
84  assert(2 == weekday1(29,2,2012)); // Wednesday
85  assert(2 == weekday1(28,11,2012)); // Wednesday
86  assert(5 == weekday1(5,1,2013)); // Saturday
87  assert(3 == weekday1(7,3,2013)); // Thursday
88  assert(6 == weekday1(24,12,2017)); // Sunday
89  assert(5 == weekday1(8,1,2000)); // Saturday
90  assert(0 == weekday1(28,2,2000)); // Monday
91  assert(5 == weekday1(11,11,2000)); // Saturday
92  assert(0 == weekday1(19,2,1900)); // Monday
93  assert(2 == weekday1(1,8,1900)); // Wednesday
94  assert(0 == weekday1(28,2,1600)); // Monday
95  assert(2 == weekday1(27,9,1600)); // Wednesday
96  assert(2 == weekday1(1,3,1600)); // Wednesday
97  assert(5 == weekday1(27,2,2100)); // Saturday
98  assert(2 == weekday1(31,3,2100)); // Wednesday
99
100
101
102
103
104  return 0;
105
106 }
107
108
109
110
111
112
113
114 // functions of my own
115
116 //Function to determine weekday,
117 // using wrong implementation using truncating division:
118 int weekday2001(int day, int month, int year){
119     //// what day was January 1st of the given year?
120     // => taking into account leap year rules
121     int yearsPassed = year - 2001;

```

```

122 int weekdayJanuary1 = (365*yearsPassed +
123     yearsPassed/4 - yearsPassed/100 + yearsPassed/400) % 7;
124
125 //Determine if given year is leap year:
126 //leap year if divisible by 4 but not by 100, or if divisible by 400
127 bool LeapYear = ((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0);
128
129
130 //How many days have passed since January 1st?
131 int daysPassed = 0;
132 (month == 1) // January?
133     ? daysPassed = day
134     : (month == 2) // February?
135       ? daysPassed = day + 31
136       : (month > 2 && LeapYear) // something else and leap year?
137         ? daysPassed = day + 60 + (153*month - 457)/5
138         : daysPassed = day + 59 + (153*month - 457)/5;
139
140 //Calculation of weekday
141 int weekday = (weekdayJanuary1 + daysPassed - 1) % 7;
142
143
144 return weekday;
145 }
146
147 //Function to determine absolute value of integer
148 int abs(int a){
149     int result = 0;
150     (a >= 0)
151         ? result = a
152         : result = -a;
153     return result;
154 }
155
156 // Function to determine sign of integer
157 int sgn(int a){
158     int result = 0;
159     (a > 0)
160         ? result = 1
161         : (a < 0)
162           ? result = -1
163           : result = 0;
164     return result;
165 }
166
167 // Function to round up
168 int floorDiv(int a, int b){
169     // Find out if result is negative (use truncating
170     //division if not)
171     int result = 0;
172     (sgn(a) == sgn(b))
173         ? result = a/b
174         : result = (-abs(a) - abs(b))/abs(b); //subtract 1 from fraction
175                                           //and round up
176     return result;
177 }
178
179 // Function to calculate the remainder taking into account floorDiv
180 int floorMod(int a, int b){
181     int result = a - b*floorDiv(a,b);
182     return result;
183 }
184
185
186 // Function to determine weekday correctly
187 int weekday(int day, int month, int year){

```

```

188  /// what day was January 1st of the given year?
189  // => taking into account leap year rules
190  int yearsPassed = year - 2001;
191  int weekdayJanuary1 = floorMod((365*yearsPassed +
192      floorDiv(yearsPassed,4) - floorDiv(yearsPassed,100)
193      + floorDiv(yearsPassed,400)),7);
194
195  ///Determine if given year is leap year:
196  //leap year if divisible by 4 but not by 100, or if divisible by 400
197  bool LeapYear = ((floorMod(year,4) == 0)
198      && (floorMod(year,100) != 0))
199      || (floorMod(year,400) == 0);
200
201
202  /// How many days have passed since January 1st?
203  int daysPassed = 0;
204  (month == 1) // January?
205      ? daysPassed = day
206      : (month == 2) // February?
207          ? daysPassed = day + 31
208          : (month > 2 && LeapYear) // something else and leap year?
209              ? daysPassed = day + 60 + floorDiv((153*month - 457),5)
210              : daysPassed = day + 59 + floorDiv((153*month - 457),5);
211
212  /// Calculation of weekday
213  int weekday = floorMod((weekdayJanuary1 + daysPassed - 1),7);
214
215
216  return weekday;
217 }
218
219 // Function for weekday without floorDiv and floorMod for subtask e)
220 int weekday1(int day, int month, int year){
221     /// what day was January 1st of the given year?
222     // => taking into account leap year rules
223     int yearsPassed = year - 1;
224     int weekdayJanuary1 = (365*yearsPassed +
225         yearsPassed/4 - yearsPassed/100 + yearsPassed/400) % 7;
226
227     ///Determine if given year is leap year:
228     //leap year if divisible by 4 but not by 100, or if divisible by 400
229     bool LeapYear = ((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0);
230
231
232     /// How many days have passed since January 1st?
233     int daysPassed = 0;
234     (month == 1) // January?
235         ? daysPassed = day
236         : (month == 2) // February?
237             ? daysPassed = day + 31
238             : (month > 2 && LeapYear) // something else and leap year?
239                 ? daysPassed = day + 60 + (153*month - 457)/5
240                 : daysPassed = day + 59 + (153*month - 457)/5;
241
242     /// Calculation of weekday
243     int weekday = (weekdayJanuary1 + daysPassed - 1) % 7;
244
245
246     return weekday;
247 }

```

Aufgabe 4

```

1 #include <iostream>
2 #include <string>

```

```

3 #include <cassert>
4 #include <cmath>
5
6 using namespace std;
7
8 //calculating square of a number
9 double sq(double x){
10     return x*x;
11 }
12
13 //recursive method to calculate powers of x
14 double power(double x, int n){
15     assert(n>0);
16     double ans;
17
18     (n==1) ?
19     ans = x :
20     ((n%2 == 0) ?
21     /*
22         if the result for n being equal would be calculated
23         as power(x,n/2)*power(x,n/2), the power function would
24         be executed twice, this is exactly what we are trying to avoid
25         to reduce the running time of the calculation, by only calculating
26         it once and using the result instead of the function itself
27         to calculate the square.
28     */
29     ans = sq(power(x,n/2)):
30     ans = x*power(x,n-1));
31     return ans;
32 }
33
34 /*
35     function that prints the relevant parameters to compare
36     the two methods of calculating powers of x
37     The comparison between our own function and std::pow was done by
38     printing the results instead of using the assert() function, because
39     of the differences described below. This would crash the program for
40     certain input values.
41 */
42 void testing(double x, int n, int i){
43     cout << "test no." << to_string(i) << ":" << endl;
44     cout << "to calculate: " << to_string(x) << "^" << to_string(n) << endl;
45     cout << "result of own function: " << to_string(power(x,n)) << endl;
46     cout << "result of std::pow(): " << to_string(std::pow(x,n)) << endl;
47     cout << "difference: " << to_string(power(x,n)-std::pow(x,n)) <<
48         endl;
49     cout << endl;
50 }
51
52 int main(){
53     testing(2,2,1);
54     testing(10,10,4);
55     testing(3.141592,10,2);
56     testing(42,13,5);
57     testing(42,42,6);
58     testing(314,100,7);
59     testing(2.71828182846,42,8);
60     testing(3.141592,42,3);
61     /*
62         interestingly the results for big powers of big or non integer
63         numbers are NOT equal. This is most likely due to the fact, that
64         the std::pow() function works with floats instead of doubles
65     */
66     return 0;
67 }

```