

Haskell Hwk2

Martin Kozeny
CSCI 4501: Programming Language Structure
Spring 2011 University of New Orleans

February 21, 2011

1 Haskell code

```
1  module Hwk2 where
2  import Data.Char
3
4  -- 1.
5  -- this function expects as input parameter list of positive numbers and
6  -- output is number; it is sum of list of numbers is counted
7  -- recursively
8  sum' :: (Num a) => [a] -> a
9  sum' [] = 0
10 sum' (x:xs) = x + sum' xs
11
12 -- this function expects as an input positive int
13 -- and output is also int; in every step counts square
14 -- of first member of the actual list and add whole sum of squares
15 -- is counted recursively
16 sumSquare :: Int -> Int
17 sumSquare 0 = 0
18 sumSquare n = n * n + sumSquare (n-1)
19
20 -- this function expects as an input positive int
21 -- and output is also int; in this example is list of squares done by the list generator
22 -- and then the sum is counted by the function sum' declared before
23 sumSquare' :: Int -> Int
24 sumSquare' n = sum' [x * x | x <- [1..n]]
25
26 -- 2.
27 -- this function expects as an input triple of positive ints
28 -- and output is Bool value; function controls if members of given triple
29 -- are sides of pythagorean triangle
30 isPyth :: (Int, Int, Int) -> Bool
31 isPyth (a, b, c) = c * c == a * a + b * b
32
33 -- this function expects as an input int and output is a list of triples;
34 -- in this function is triple of sides generated by the list generator
35 -- and it is controlled by the function isPyth if given sides are sides
36 -- of pythagorean triangle; for more efficiency it is also controlled if sides
37 -- ordinates a and b are lower than hypotenuse c and if this sides can create
38 -- a triangle
39 pyths :: Int -> [(Int, Int, Int)]
40 pyths n = [(a, b, c) | a <- [1..n], b <- [1..n], c <- [1..n], a < c, b < c,
41   (a + b > c), isPyth (a, b, c)]
42
43 -- this function expects as an input int and output is a list of triples;
44 -- in this function is triple of sides generated by the list generator
```

```

45  -- and it is controlled by the function isPyth' in where clause if given sides are sides
46  -- of pythagorean triangle; for more efficiency it is also controlled if sides
47  -- ordinates a and b are lower than hypotenuse c and if this sides can create
48  -- a tringle
49  pyths' :: Int -> [(Int, Int, Int)]
50  pyths' n = [(a, b, c) | a <- [1..n], b <- [1..n], c <- [1..n], a < c, b < c,
51    (a + b > c), isPyth' (a, b, c)]
52    where isPyth' (a, b, c) = c * c == a * a + b * b
53
54  -- 3.
55  -- this function expects as an input list of ints and int and output is a list of ints;
56  -- function computes the 'mirror' divisors to a number when are divisors to root known
57  computeRestDivisors :: [Integer] -> [Integer]
58  computeRestDivisors [] n = []
59  computeRestDivisors (x:xs) n = if ((x /= (div n x)) && (x/=1))
60    then x : ((div n x) : (computeRestDivisors xs n))
61    else if (n/=x)
62    then x : (computeRestDivisors xs n)
63    else computeRestDivisors xs n
64
65  -- this function expects as an input int and output is a list of ints;
66  -- this function counts by the list generator divisors of given number
67  -- to the root of the number and number itself is omitted
68  findDivisors :: Integer -> [Integer]
69  findDivisors num = let root = round (sqrt (fromInteger num)) in
70    computeRestDivisors [number | number <- [1..root], mod num number == 0] num
71
72  -- this function expects as an input int and output is a list of ints;
73  -- function generate list and adds concrete number if his sum of divisors
74  -- equals to number itself (uses first implementation of 'findDivisors')
75  perfectInt :: Integer -> [Integer]
76  perfectInt n = [x | x <- [1..n], sum' (findDivisors x) == x]
77
78  -- this function expects as an input int and output is a list of ints;
79  -- this function counts by the list generator divisors of given number
80  -- to the number and number itself is omitted; this is expected to be
81  -- a slower implementation than function 'findDivisors'
82  findDivisors' :: Int -> [Int]
83  findDivisors' n = [x | x <- [1..(n-1)], mod n x == 0]
84
85  -- this function expects as an input int and output is a list of ints;
86  -- function generate list and adds concrete number if his sum of divisors
87  -- equals to number itself; it is similar function like function 'perfectInt'
88  -- but for control uses second implementation of 'findDivisors'
89  perfectInt' :: Int -> [Int]
90  perfectInt' n = [x | x <- [1..n], sum' (findDivisors' x) == x]
91
92  -- 4.
93  -- this function expects as an input a list of characters and output is a tuple
94  -- where first member are numbers extracted form the list
95  -- and the second member is te rest; function recursively controls if first member
96  -- of actual list is number; if yes, then the member is added to the first member
97  -- of the tuple, if not then this member is added to the second part of the tuple;
98  -- the rest of the list is send recursively to the next step and in each step
99  -- is created a tuple and actual member is added to only a one part(first or second)
100  -- of the tuple; when is resulted tuple constructed, it is only taken into account
101  -- the member of the tuple according to his actual position (first or second)
102  splitString :: [Char] -> ([Char],[Char])
103  splitString [] = ([], [])
104  splitString (x:xs) = if isNumber x
105    then (x: (fst (splitString xs)), snd (splitString xs))
106    else (fst (splitString xs), x: (snd (splitString xs)) )

```

```

107
108
109 -- 5.
110 -- this function expects as an input a two lists of 'orderable' symbols
111 -- and output is ordered list of this symbols; the Ord class is used for
112 -- totally ordered datatypes; the Ordering datatype allows a single comparison
113 -- to determine the precise ordering of two objects(it is probably similar
114 -- like in Java implements comparable); this function is used for joining
115 -- two ordered lists into ordered list.
116 merge :: Ord a => [a] -> [a] -> [a]
117 merge [] [] = []
118 merge [] (x:[]) = [x]
119 merge (x:[]) [] = [x]
120 merge (x:[]) (y:[]) = if x <= y
121                       then [x] ++ [y]
122                       else
123                         [y] ++ [x]
124 merge (x:xs) (y:[]) = if x <= y
125                       then x : merge xs [y]
126                       else
127                         y : (x:xs)
128 merge (x:[]) (y:ys) = if y <= x
129                       then y : merge ys [x]
130                       else
131                         x : (y:ys)
132
133 merge (x:xs) (y:ys) = if x <= y
134                       then x : merge xs (y:ys)
135                       else
136                         y : merge (x:xs) ys
137
138 -- this function expects as an input a two lists of 'orderable' symbols
139 -- and output is ordered list of this symbols; Here is also used Ord class
140 -- explained before; this function firstly recursively divide input list
141 -- in halves and then these peices are joined and ordered by function merge
142 mergeSort :: Ord a => [a] -> [a]
143 mergeSort [] = []
144 mergeSort (x:[]) = [x]
145 mergeSort xs = merge (mergeSort (fst (splitAt (div (length xs) 2) xs)))
146                   (mergeSort (snd (splitAt (div (length xs) 2) xs)))
147
148 -- 6.
149 -- a.
150 -- this function expects as an input orderable symbol and list of 'orderable' symbols
151 -- and output is boolean value; this function only tests, if given list contains
152 -- input symbol
153 containList :: Ord a => a -> [a] -> Bool
154 containList x [] = False
155 containList x (y:ys) = if x == y
156                       then True
157                       else containList x ys
158
159 -- this function expects as an input two lists of 'orderable' symbols
160 -- and output is boolean value; this function tests if members from the first list
161 -- are contained in the second list; it takes one member after another and tests it;
162 -- when find first not contained member, then return False, otherwise when he reaches
163 -- empty first list, then returns True
164 containElements :: Ord a => [a] -> [a] -> Bool
165 containElements [] ys = True
166 containElements (x:xs) ys = if containList x ys
167                             then containElements xs ys
168                             else

```

```

169                                     False
170
171  -- b.
172  -- this function expects as an input two lists of 'orderable' symbols
173  -- and output is boolean value; this function tests if members from first
174  -- list are contained as a subsequence in the second list; it is controlled one after
175  -- another member of the first list with actual rest of the second list;
176  -- if function reaches end of the first list despite second list reaches its end
177  -- or not returns True; if some members remains in the first list, then returns False;
178  subsequence :: Ord a => [a] -> [a] -> Bool
179  subsequence [] (y:ys) = True
180  subsequence [] [] = True
181  subsequence (x:xs) [] = False
182  subsequence (x:xs) (y:ys) = if x == y
183                               then subsequence xs ys
184                               else subsequence (x:xs) ys

```

2 Test script

```

1  -- 1. Define a function that computes the sum of the of the squares 1..n, given n\r\n");
2
3  sumSquare 5
4  -- expected 55
5
6  sumSquare 8
7  -- expected 204
8
9  sumSquare 10
10 -- expected 385
11
12 sumSquare 13
13 -- expected 819
14
15 sumSquare 15
16 -- expected 1240
17
18 -- another option of sumSquare
19
20 sumSquare' 5
21 -- expected 55
22
23 sumSquare' 8
24 -- expected 204
25
26 sumSquare' 10
27 -- expected 385
28
29 sumSquare' 13
30 -- expected 819
31
32 sumSquare' 15
33 -- expected 1240
34
35 -- 2. Define the function that returns the list of all pythagorean triples
36 -- whose components greater than 0 and less than the given integer.
37
38 pyths 10
39
40 -- expected
41 -- [(3,4,5),(4,3,5),(6,8,10),(8,6,10)]

```

```

42
43 pyths 20
44
45 --expected
46 --[(3,4,5),(4,3,5),(5,12,13),(6,8,10),(8,6,10),(8,15,17),(9,12,15),
47 --(12,5,13),(12,9,15),(12,16,20),(15,8,17),(16,12,20)]
48
49 pyths 30
50
51 --expected
52 --[(3,4,5),(4,3,5),(5,12,13),(6,8,10),(7,24,25),(8,6,10),(8,15,17),
53 --(9,12,15),(10,24,26),(12,5,13),(12,9,15),(12,16,20),(15,8,17),(15,20,25),
54 --(16,12,20),(18,24,30),(20,15,25),(20,21,29),(21,20,29),(24,7,25),
55 --(24,10,26),(24,18,30)]
56
57 pyths 40
58
59 --expected
60 --[(3,4,5),(4,3,5),(5,12,13),(6,8,10),(7,24,25),(8,6,10),(8,15,17),(9,12,15),
61 --(10,24,26),(12,5,13),(12,9,15),(12,16,20),(12,35,37),(15,8,17),(15,20,25),
62 --(15,36,39),(16,12,20),(16,30,34),(18,24,30),(20,15,25),(20,21,29),(21,20,29),
63 --(21,28,35),(24,7,25),(24,10,26),(24,18,30),(24,32,40),(28,21,35),(30,16,34),
64 --(32,24,40),(35,12,37),(36,15,39)]
65
66 pyths 50
67
68 --expected
69 --[(3,4,5),(4,3,5),(5,12,13),(6,8,10),(7,24,25),(8,6,10),(8,15,17),(9,12,15),
70 --(9,40,41),(10,24,26),(12,5,13),(12,9,15),(12,16,20),(12,35,37),(14,48,50),
71 --(15,8,17),(15,20,25),(15,36,39),(16,12,20),(16,30,34),(18,24,30),(20,15,25),
72 --(20,21,29),(21,20,29),(21,28,35),(24,7,25),(24,10,26),(24,18,30),(24,32,40),
73 --(27,36,45),(28,21,35),(30,16,34),(30,40,50),(32,24,40),(35,12,37),(36,15,39),
74 --(36,27,45),(40,9,41),(40,30,50),(48,14,50)]
75
76 -- another option of pyths
77
78 pyths' 10
79
80 --expected
81 --[(3,4,5),(4,3,5),(6,8,10),(8,6,10)]
82
83 pyths' 20
84
85 --expected
86 --[(3,4,5),(4,3,5),(5,12,13),(6,8,10),(8,6,10),(8,15,17),(9,12,15),
87 --(12,5,13),(12,9,15),(12,16,20),(15,8,17),(16,12,20)]
88
89 pyths' 30
90
91 --expected
92 --[(3,4,5),(4,3,5),(5,12,13),(6,8,10),(7,24,25),(8,6,10),(8,15,17),
93 --(9,12,15),(10,24,26),(12,5,13),(12,9,15),(12,16,20),(15,8,17),(15,20,25),
94 --(16,12,20),(18,24,30),(20,15,25),(20,21,29),(21,20,29),(24,7,25),
95 --(24,10,26),(24,18,30)]
96
97 pyths' 40
98
99 --expected
100 --[(3,4,5),(4,3,5),(5,12,13),(6,8,10),(7,24,25),(8,6,10),(8,15,17),(9,12,15),
101 --(10,24,26),(12,5,13),(12,9,15),(12,16,20),(12,35,37),(15,8,17),(15,20,25),
102 --(15,36,39),(16,12,20),(16,30,34),(18,24,30),(20,15,25),(20,21,29),(21,20,29),
103 --(21,28,35),(24,7,25),(24,10,26),(24,18,30),(24,32,40),(28,21,35),(30,16,34),

```

```

104  -- (32,24,40),(35,12,37),(36,15,39)]
105
106  pyths' 50
107
108  -- expected
109  -- [(3,4,5),(4,3,5),(5,12,13),(6,8,10),(7,24,25),(8,6,10),(8,15,17),(9,12,15),
110  -- (9,40,41),(10,24,26),(12,5,13),(12,9,15),(12,16,20),(12,35,37),(14,48,50),
111  -- (15,8,17),(15,20,25),(15,36,39),(16,12,20),(16,30,34),(18,24,30),(20,15,25),
112  -- (20,21,29),(21,20,29),(21,28,35),(24,7,25),(24,10,26),(24,18,30),(24,32,40),
113  -- (27,36,45),(28,21,35),(30,16,34),(30,40,50),(32,24,40),(35,12,37),(36,15,39),
114  -- (36,27,45),(40,9,41),(40,30,50),(48,14,50)]
115
116  -- 3. A positive integer is perfect if it equals the sum of its factors, excluding the number itself.
117  -- Define the function that returns the list of all perfect numbers up to the given input number
118
119  perfectInt 30
120
121  -- expected
122  -- [6,28]
123
124  perfectInt 100
125
126  -- expected
127  -- [6,28]
128
129  perfectInt 500
130
131  -- expected
132  -- [6,28,496]
133
134  perfectInt 1000
135
136  -- expected
137  -- [6,28,496]
138
139  perfectInt 10000
140
141  -- expected
142  -- [6,28,496,8128]
143
144  -- another option of perfectInt
145
146  perfectInt' 30
147
148  -- expected
149  -- [6,28]
150
151  perfectInt' 100
152
153  -- expected
154  -- [6,28]
155
156  perfectInt' 500
157
158  -- expected
159  -- [6,28,496]
160
161  perfectInt' 1000
162
163  -- expected
164  -- [6,28,496]
165

```

```

166 perfectInt' 10000
167
168 --expected
169 --[6,28,496,8128]
170
171
172 --4. Define a function that takes a string containing digits and any other characters, and returns a pair
173 -- where the first component is the list of digits in the string in the order they occur in the input string,
174 -- and the second pair contains the rest of characters of the string in the order they occur.
175
176 splitString "Value#1 value#2 35 !"
177
178 --expected
179 --("1235","Value# value# !")
180
181 splitString "GreAT &&*30!0 FuN B9o0Y!23#"
182
183 --expected
184 --("23009023","GreAT &&*! FuN BoY!#")
185
186 splitString "l23& 1jU1**s12T CA#32me T98#2O s#2Ay# He43LL2o!#?#"
187
188 --expected
189 --("231112329822432","l& jU**sT CA#me T#O s#Ay# HeLLo!#?#")
190
191 splitString "T56h44e65 O656nL4y4 W65A245y32 T32o32 HA23vE 2a34 FR#i$En$d I5S3 t[$$]3452O B76e 3A11 FRiE_32nD!(23)"
192
193 --expected
194 --("564465656446524532323223234533452763113223","The OnLy WAy To HAvE a FR#i$En$d IS t[$$]O Be A FRiE_nD!()")
195
196 --5. Define merge sort.
197
198 mergeSort [2,12,213,45,786,221,13,56,67,64,203]
199
200 --expected
201 --[2,12,13,45,56,64,67,203,213,221,786]
202
203 mergeSort [43,123,321,25,76,211,131,12]
204
205 --expected
206 --[12,25,43,76,123,131,211,321]
207
208 mergeSort [423,13,31,225,76,21,1231,412,152,2,382,341,65,586,23]
209
210 --expected
211 --[2,13,21,23,31,65,76,152,225,341,382,412,423,586,1231]
212
213 --6. (only for Haskell).
214 --a. Define a function that takes two lists list1 and list2 and returns true if list1s elements are in list2.
215
216 containElements [1,2,5,4,3,4,5,8,7] [1,2,3,4,5,6,7,8,9,0]
217
218 --expected
219 --True
220
221 containElements [9,9,4,6,6,6,11] [1,2,3,4,5,6,7,8,9,0]
222
223 --expected
224 --False
225
226 containElements [22,34,5,4,3,2,1] [1,2,3,4,5,6,7,8,9,0,34,22]
227

```

```

228 --expected
229 -- True
230
231 containElements [2,5,7,9] [1,2,3,4,5,6,7,8,9,0]
232
233 --expected
234 -- True
235
236 --b. Define a function that takes two lists list1 and list2 and returns true if list1 appears in list2 as a subsequence. To
237 --appear as a subsequence means that the elements in list1 appear in the same sequence in list2.
238
239 subsequence [22,34,5,4,3,2,1] [1,2,3,4,5,6,7,8,9,0,34,22]
240
241 --expected
242 -- False
243
244 subsequence [2,5,7,9] [1,2,3,4,5,6,7,8,9,0]
245
246 --expected
247 -- True
248
249 subsequence [12,15,17,22,35,71] [12,13,14,15,16,17,18,19,20,21,22,33,34,35,76,77,78]
250
251 --expected
252 -- False
253
254 subsequence [12,15,17,22,35,77] [12,13,14,15,16,17,18,19,20,21,22,33,34,35,76,77,78]
255
256 --expected
257 -- True
258
259 subsequence [1,5,7,9,12] [1,3,4,5,7,8,9,11,12,13]
260
261 --expected
262 -- True
263
264 subsequence [1,5,7,9,12] [1,10,9,5,7,8,11,12,13]
265
266 --expected
267 -- False

```

3 C code

```

1 #include <stdio.h>
2 #include <string.h>
3 #include "headers/node.h"
4 #include "headers/tuple.h"
5 #include "headers/functions.h"
6
7 int main() {
8     printf("1. Define a function that computes the sum of the of the squares 1..n, given n\r\n");
9     int n = 5;
10    printf("for n=%i is sum of squares: %i\r\n", n, sum_square(n));
11    n = 8;
12    printf("for n=%i is sum of squares: %i\r\n", n, sum_square(n));
13    n = 10;
14    printf("for n=%i is sum of squares: %i\r\n", n, sum_square(n));
15    n = 13;
16    printf("for n=%i is sum of squares: %i\r\n", n, sum_square(n));
17    n = 15;

```



```

18     printf("for n=%i is sum of squares: %i\r\n", n, sum_square(n));
19
20     struct NODE *llist;
21     llist = (struct NODE *)malloc(sizeof(struct NODE));
22     printf("2. Define the function that returns the list of all pythagorean triples "
23           "whose components greater than 0 and less that the given integer.\r\n");
24     n=10;
25     printf("for n=%i are all pythagorean triples:\r\n", n);
26     is_pyth(n, llist);
27     display_list_array(llist, 3);
28     llist = (struct NODE *)malloc(sizeof(struct NODE));
29     n=20;
30     printf("for n=%i are all pythagorean triples:\r\n", n);
31     is_pyth(n, llist);
32     display_list_array(llist, 3);
33     llist = (struct NODE *)malloc(sizeof(struct NODE));
34     n=30;
35     printf("for n=%i are all pythagorean triples:\r\n", n);
36     is_pyth(n, llist);
37     display_list_array(llist, 3);
38     llist = (struct NODE *)malloc(sizeof(struct NODE));
39     n=40;
40     printf("for n=%i are all pythagorean triples:\r\n", n);
41     is_pyth(n, llist);
42     display_list_array(llist, 3);
43     llist = (struct NODE *)malloc(sizeof(struct NODE));
44     n=50;
45     printf("for n=%i are all pythagorean triples:\r\n", n);
46     is_pyth(n, llist);
47     display_list_array(llist, 3);
48
49     llist = (struct NODE *)malloc(sizeof(struct NODE));
50     printf("3. A positive integer is perfect if it equals the sum of its factors, excluding the number itself. Define the function "
51           "that returns the list of all perfect numbers up to the given input number\r\n");
52     n=30;
53     perfectInt(n, llist);
54     printf("for n=%i are perfect numbers:\r\n", n);
55     display_list(llist);
56     llist = (struct NODE *)malloc(sizeof(struct NODE));
57     n=100;
58     perfectInt(n, llist);
59     printf("for n=%i are perfect numbers:\r\n", n);
60     display_list(llist);
61     llist = (struct NODE *)malloc(sizeof(struct NODE));
62     n=500;
63     perfectInt(n, llist);
64     printf("for n=%i are perfect numbers:\r\n", n);
65     display_list(llist);
66     llist = (struct NODE *)malloc(sizeof(struct NODE));
67     n=1000;
68     perfectInt(n, llist);
69     printf("for n=%i are perfect numbers:\r\n", n);
70     display_list(llist);
71     llist = (struct NODE *)malloc(sizeof(struct NODE));
72     n=10000;
73     perfectInt(n, llist);
74     printf("for n=%i are perfect numbers:\r\n", n);
75     display_list(llist);
76
77     struct TUPLE * tuple;
78     printf("4. Define a function that takes a string containing digits and any other characters, and returns a pair where the first "
79           "component is the list of digits in the string in the order they occur in the input string, and the second pair contains the "

```

```

80         "rest of characters of the string in the order they occur.\r\n");
81 tuple = (struct TUPLE *)malloc(sizeof(struct TUPLE));
82 char * str ="Value#1 value#2 35 !";
83 init_tuple(tuple, str);
84 splitString(str, tuple);
85 printf("for \"%s\" is requested tuple:\r\n", str);
86 print_tuple(tuple);
87 tuple = (struct TUPLE *)malloc(sizeof(struct TUPLE));
88 str ="GreAT &&2*30!0 FuN B9o0Y!23#";
89 init_tuple(tuple, str);
90 splitString(str, tuple);
91 printf("for \"%s\" is requested tuple:\r\n", str);
92 print_tuple(tuple);
93 tuple = (struct TUPLE *)malloc(sizeof(struct TUPLE));
94 str ="I23& 1jU1**s12T CA#32me T98#2O s#2Ay# He43LL2o!#?#";
95 init_tuple(tuple, str);
96 splitString(str, tuple);
97 printf("for \"%s\" is requested tuple:\r\n", str);
98 print_tuple(tuple);
99 tuple = (struct TUPLE *)malloc(sizeof(struct TUPLE));
100 str ="T56h44e65 O656nL4y4 W65A245y32 T32o32 HA23vE 2a34 FR#i$En$d I5S3 t[$$]3452O B76e 3A11 FRiE_32nD!(23)";
101 init_tuple(tuple, str);
102 splitString(str, tuple);
103 printf("for \"%s\" is requested tuple:\r\n", str);
104 print_tuple(tuple);
105
106 printf("5. Define merge sort.\r\n");
107 int length = 11;
108 int * list = malloc(length*sizeof(int));
109 list[0]=2;
110 list[1]=12;
111 list[2]=213;
112 list[3]=45;
113 list[4]=786;
114 list[5]=221;
115 list[6]=13;
116 list[7]=56;
117 list[8]=67;
118 list[9]=64;
119 list[10]=203;
120 printf("For unsorted array:\r\n");
121 array_output(list, length);
122 int * sorted_list = merge_sort(list, length);
123 printf("is sorted array:\r\n");
124 array_output(sorted_list, length);
125 length = 8;
126 list = malloc(length*sizeof(int));
127 list[0]=43;
128 list[1]=123;
129 list[2]=321;
130 list[3]=25;
131 list[4]=76;
132 list[5]=211;
133 list[6]=131;
134 list[7]=12;
135 printf("For unsorted array:\r\n");
136 array_output(list, length);
137 sorted_list = merge_sort(list, length);
138 printf("is sorted array:\r\n");
139 array_output(sorted_list, length);
140 length = 15;
141 list = malloc(length*sizeof(int));

```

```

142     list[0]=423;
143     list[1]=13;
144     list[2]=31;
145     list[3]=225;
146     list[4]=76;
147     list[5]=21;
148     list[6]=1231;
149     list[7]=412;
150     list[8]=152;
151     list[9]=2;
152     list[10]=382;
153     list[11]=341;
154     list[12]=65;
155     list[13]=586;
156     list[14]=23;
157     printf("For unsorted array:\r\n");
158     array_output(list, length);
159     sorted_list = merge_sort(list, length);
160     printf("is sorted array:\r\n");
161     array_output(sorted_list, length);
162
163     free(llist);
164     free(tuple);
165     free(list);
166     free(sorted_list);
167
168     return 0;
169 }
170 /* input of this function is int ad this function also returns int
171    * in the for cycle counts square of number 1 to n and add the result to sum
172    */
173 int sum_square(int n) {
174     int i;
175     int sum = 0;
176     for (i = 1; i <= n; i++)
177         sum+=i*i;
178     return sum;
179 }
180 /* this function expects as an input int and NODE pointer
181    * and output is a list of triples; in this function are three sides of triangle
182    * controlled by the function test_pyth if given sides are sides
183    * of pythagorean triangle; if yes they are add to the linked list
184    * of NODEs in array
185    */
186 void is_pyth(int n, struct NODE *llist) {
187     int* result;
188     int c;
189     int a;
190     int b;
191     for (c = 1; c <= n; c++) {
192         for (a = 1; a < c; a++) {
193             for (b = 1; b < c; b++) {
194                 if (test_pyth(a, b, c)) {
195                     result = malloc(3 * sizeof(int));
196                     result[0] = a;
197                     result[1] = b;
198                     result[2] = c;
199                     append_node_array(llist, result);
200                 }
201             }
202         }
203     }

```

```

204 }
205 /* this function expects as input three ints and output is also int
206 * function controls if input values
207 * are sides of pythagorean triangle
208 * */
209 int test_pyth(int a, int b, int c) {
210     if (a*a + b*b == c*c)
211         return 1;
212     else
213         return 0;
214 }
215 /* this function expects as an input int and a linked list of NODEs;
216 * firstly function get list of divisors, then sum them and tests if
217 * the result is equal to actual number, if yes actual number is added
218 * */
219 void perfectInt(int n, struct NODE *llist) {
220     struct NODE *llist1;
221     int i;
222     llist1 = (struct NODE *)malloc(sizeof(struct NODE));
223     for (i = 1; i <= n; i++) {
224         divisors(i, llist1);
225         if (sum(llist1)==i) {
226             append_node(llist, i);
227         }
228         llist1 = (struct NODE *)malloc(sizeof(struct NODE));
229     }
230     free(llist1);
231 }
232 /* this function expects as an input int and a linked list of NODEs;
233 * function fills input linked list with divisors of input integer n
234 * except for number itself
235 * */
236 void divisors(int n, struct NODE *llist1) {
237     int i;
238     for (i = 1; i < n; i++) {
239         if ((n % i) == 0)
240             append_node(llist1, i);
241     }
242 }
243 /* this function expects as an input array of characters and a TUPLE;
244 * function fills in cycle first part of input TUPLE with integers from input
245 * char array and second part fills with rest characters
246 * */
247 void splitString(char* s, struct TUPLE *tuple) {
248     int i;
249     int length = strlen(s);
250     for (i = 0; i < length; i++) {
251         if (s[i]>=48 && s[i]<=57)
252             add_number_to_tuple(tuple, s[i]);
253         else
254             add_character_to_tuple(tuple, s[i]);
255     }
256 }
257 /*this function expects as an input array of ints and one integer
258 * (length of that array)
259 * this function firstly recursively divide input list
260 * in halves and then these peices are joined and ordered by function merge
261 * */
262 int* merge_sort(int* list, int length) {
263     if (length <= 1)
264         return list;
265     int * left;

```

```

266     int * right;
267     int * result;
268     int middle = length / 2;
269     left = malloc(middle*sizeof(int));
270     right = malloc((length-middle)*sizeof(int));
271     result = malloc(length*sizeof(int));
272     int i;
273     for (i = 0; i < middle; i++)
274         left[i] = list[i];
275     for (i = middle; i < length; i++)
276         right[i-middle] = list[i];
277
278     left = merge_sort(left, middle);
279     right = merge_sort(right, (length - middle));
280     result = merge(left, right, middle, (length - middle));
281     free(left);
282     free(right);
283     return result;
284 }
285 /* this function expects as an input two arrays of ints and two integers
286 * (lengths of that arrays)
287 * this function is used for joining two ordered lists into ordered list
288 * */
289 int* merge(int* left, int* right, int lengthLeft, int lengthRight) {
290     int* result = malloc((lengthLeft+lengthRight)*sizeof(int));
291     int resultPointer=0;
292     int leftPointer=0;
293     int rightPointer=0;
294     while (lengthLeft > leftPointer || lengthRight > rightPointer) {
295         if (lengthLeft > leftPointer && lengthRight > rightPointer) {
296             if (left[leftPointer] <= right[rightPointer]) {
297                 result[resultPointer] = left[leftPointer];
298                 resultPointer++;
299                 leftPointer++;
300             } else {
301                 result[resultPointer] = right[rightPointer];
302                 resultPointer++;
303                 rightPointer++;
304             }
305         } else if (lengthLeft > leftPointer) {
306             result[resultPointer] = left[leftPointer];
307             resultPointer++;
308             leftPointer++;
309         } else if (lengthRight > rightPointer) {
310             result[resultPointer] = right[rightPointer];
311             resultPointer++;
312             rightPointer++;
313         }
314     }
315     return result;
316 }
317
318 void array_output(int* list, int length) {
319     int i;
320     for (i = 0; i < length; i++)
321         printf("%i ", list[i]);
322     printf("\r\n");
323 }

```

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "headers/node.h"
4
5  void display_list(struct NODE *llist) {
6      if (llist->next != NULL)
7          llist = llist->next;
8      while (llist->next != NULL) {
9          printf("%d ", llist->number);
10         llist = llist->next;
11     }
12     printf("%d \r\n", llist->number);
13 }
14 void display_list_array(struct NODE *llist, int length) {
15     if (llist->next != NULL)
16         llist = llist->next;
17     int i;
18     while (llist->next != NULL) {
19
20         for (i = 0; i < length; i++) {
21             printf("%i ", llist->array[i]);
22         }
23         printf("\r\n");
24         llist = llist->next;
25     }
26     if (llist->array != NULL) {
27         for (i = 0; i < length; i++) {
28             printf("%i ", llist->array[i]);
29         }
30         printf("\r\n");
31     }
32 }
33 void append_node(struct NODE *llist, int num) {
34     while (llist->next != NULL)
35         llist = llist->next;
36
37     llist->next = (struct NODE *)malloc(sizeof(struct NODE));
38     llist->next->number = num;
39     llist->next->next = NULL;
40 }
41 void append_node_array(struct NODE *llist, int* array) {
42     while (llist->next != NULL)
43         llist = llist->next;
44
45     llist->next = (struct NODE *)malloc(sizeof(struct NODE));
46     llist->next->array = array;
47     llist->next->next = NULL;
48 }
49 int sum(struct NODE *llist) {
50     int sum;
51     if (llist->next != NULL)
52         llist = llist->next;
53     while (llist->next != NULL) {
54         sum += llist->number;
55         llist = llist->next;
56     }
57     sum += llist->number;
58     return sum;
59 }

```

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #include "headers/tuple.h"
6
7  void init_tuple(struct TUPLE *tuple, char* str) {
8      tuple->numbers = malloc(strlen(str) * sizeof(char));
9      tuple->characters = malloc(strlen(str) * sizeof(char));;
10     tuple->pointerNumbers = 0;
11     tuple->pointerCharacters = 0;
12 }
13 void add_number_to_tuple(struct TUPLE *tuple, char num) {
14     tuple->numbers[tuple->pointerNumbers] = num;
15     tuple->pointerNumbers++;
16 }
17 void add_character_to_tuple(struct TUPLE *tuple, char c) {
18     tuple->characters[tuple->pointerCharacters] = c;
19     tuple->pointerCharacters++;
20 }
21 void print_tuple(struct TUPLE *tuple) {
22     printf("(\\\"%s\\\", \\\"%s\\\")", tuple->numbers, tuple->characters);
23     printf("\\r\\n");
24 }
```
