Haskell Hwk2

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

February 21, 2011

1 Haskell code

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

```
— and it is controlled by the function isPyth' in where clause if given sides are sides
     — of pythagorean triangle; for more efficiency it is also controlled if sides
46
     -- ordinates a and b are lower than hypotenuse c and if this sides can create
47
     — a tringle
48
     pyths' :: Int -> [(Int, Int, Int)]
49
     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)]
         where isPyth' (a, b, c) = c * c == a * a + b * b
52
53
54
     — this function expects as an input list of ints and int and output is a list of ints;
55
     — function computes the 'mirror' divisors to a number when are divisors to root known
56
57
     computeRestDivisors :: [Integer] -> Integer -> [Integer]
     computeRestDivisors [] [] []
58
     computeRestDivisors (x:xs) n = if((x /= (div n x)) \&\& (x/=1))
59
                                        then x : ((div n x) : (computeRestDivisors xs n))
60
                                       else if (n/=x)
61
                                       then x : (computeRestDivisors xs n)
62
                                       else computeRestDivisors xs n
63
     — this function expects as an input int and output is a list of ints;
65
     — this function counts by the list generator divisors of given number
66
     -- to the root of the number and number itself is omitted
67
     findDivisors :: Integer -> [Integer]
     findDivisors num = let root = round ( sqrt (fromInteger num)) in
69
                      computeRestDivisors [number | number < [1..root], mod num number == 0] num
70
     — this function expects as an input int and output is a list of ints;
72
     — function generate list and adds concrete number if his sum of divisors
73

    equals to number itself (uses first implementation of 'findDivisors')

74
     perfectInt :: Integer -> [Integer]
75
     perfectInt n = [x | x < -[1..n], sum'(findDivisors x) = =x]
     — this function expects as an input int and output is a list of ints;
78
     — this function counts by the list generator divisors of given number
79
     — to the number and number itself is omitted; this is expected to be
80
     — a slower implementation than function 'findDivisors'
     findDivisors' :: Int -> [Int]
82
     find Divisors' n = [x | x < -[1..(n-1)], mod | n x == 0]
     — this function expects as an input int and output is a list of ints;
85

    function generate list and adds concrete number if his sum of divisors

86
     — equals to number itself; it is similar function like function 'perfectInt'
87
     — but for control uses second implementation of 'findDivisors'
88
     perfectInt' :: Int -> [Int]
89
     perfectInt' n = [x | x < -[1..n], sum'(findDivisors' x) == x]
91
92
     — this function expects as an input a list of characters and output is a tuple
93
     -- where first member are numbers extracted form the list
94
     -- and the second member is te rest; function recusrsively controls if first member
     — of actual list is number; if yes, then the member is added to the first member
     — of the tuple, if not then this member is added to the second part of the tuple;
     — the rest of the list is send recursively to the next step and in each step
     — is created a tuple and actual member is added to only a one part(first or second)
99
     — of the tuple; when is resulted tuple constructed, it is only taken into account
100
     —— the member of the tuple according to his actual position (first or second)
101
     splitString :: [Char] -> ([Char],[Char])
102
     splitString [] = ([], [])
     splitString(x:xs) = if isNumber x
104
                              then (x: (fst (splitString xs)), snd (splitString xs))
105
                              else (fst (splitString xs), x: (snd (splitString xs)) )
106
```

```
107
108
109
     — this function expects as an input a two lists of 'orderable' symbols
110
     — and output is ordered list of this symbols; the Ord class is used for
111
     -- totally ordered datatypes; the Ordering datatype allows a single comparison
         to determine the precise ordering of two objects(it is probably similar

    like in Java implements comparable); this function is used for joining

114
         two ordered lists into ordered list.
115
     merge :: Ord a => [a] -> [a] -> [a]
116
     merge [] [] = []
117
     merge [](x:[]) = [x]
     merge (x:[]) [] = [x]
119
     merge (x:[]) (y:[]) = if x \le y
120
                               then [x] ++ [y]
121
                             else
122
                               [y] ++ [x]
123
     merge (x:xs) (y:[]) = if x \le y
124
                               then x: merge xs [y]
125
                             else
126
                               y: (x:xs)
127
     merge (x:[]) (y:ys) = if y \le x
128
                               then y: merge ys [x]
129
130
                               x:(y:ys)
131
132
     merge (x:xs) (y:ys) = if x \le y
133
                               then x : merge xs (y:ys)
134
135
                               y: merge (x:xs) ys
136
137
     — this function expects as an input a two lists of 'orderable' symbols
         and output is ordered list of this symbols; Here is also used Ord class
     — explained before; this function firstly recursively divide input list
140
     — in halves and then these peices are joined and ordered by function merge
141
     mergeSort :: Ord a => [a] -> [a]
142
     mergeSort [] = []
143
     mergeSort(x:[]) = [x]
144
     mergeSort xs = merge (mergeSort (fst (splitAt (div (length xs) 2) xs)))
                        (mergeSort (snd (splitAt (div (length xs) 2) xs)))
147
     -- 6.
148
149
     — this function expects as an input orderable symbol and list of 'orderable' symbols
150
     -- and output is boolean value; this function only tests, if given list contains
151
     -- input symbol
     containList :: Ord a => a -> [a] -> Bool
153
     containList \times [] = False
154
     containList x (y:ys) = if x == y
155
                               then True
156
                              else containList x ys
157
     -- this function expects as an input two lists of 'orderable' symbols
     — and output is boolean value; this function tests if members from the first list
160
         are contained in the second list; it takes one member after another and tests it;
161
         when find first not contained member, then return False, otherwise when he reaches
162
         empty first list, then returns True
163
     containElements :: Ord a => [a] -> [a] -> Bool
164
     containElements [] ys = True
     containElements (x:xs) ys = if containList x ys
166
                                     then containElements xs ys
167
                                    else
168
```

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

2 Test script

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

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

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

```
perfectInt' 10000
166
167
     —expected
168
     --[6,28,496,8128]
169
170
171
     --4. Define a function that takes a string containing digits and any other characters, and returns a pair
172
     — where the first component is the list of digits in the string in the order they occur in the input string,
173
     — and the second pair contains the rest of characters of the string in the order they occur.
174
175
     splitString "Value#1 value#2 35!"
176
177
178
     --expected
     --("1235","Value# value# !")
179
180
     splitString "GreAT &&2*30!0 FuN B9o0Y!23#"
181
182
     --expected
183
     --("23009023","GreAT &&*! FuN BoY!#")
184
185
     splitString "I23& 1jU1**s12T CA#32me T98#2O s#2Ay# He43LL2o!#?#"
186
187
     --expected
188
     --("231112329822432"," | iU**sT CA#me T#O s#Ay# HeLLo!#?#")
189
     splitString "T56h44e65 O656nL4y4 W65A245y32 T32o32 HA23vE 2a34 FR#i$En$d I5S3 t[$$]3452O B76e 3A11 FRiE_32nD!(23)"
191
192
     --expected
193
     −−("5644656564465245323232323234533452763113223","The OnLy WAy To HAvE a FR#i$En$d IS t[$$]O Be A FRiE_nD!()")
194
195
     −−5. Define merge sort.
196
     mergeSort [2,12,213,45,786,221,13,56,67,64,203]
198
199
200
     --[2,12,13,45,56,64,67,203,213,221,786]
201
202
     mergeSort [43,123,321,25,76,211,131,12]
203
204
     --expected
205
     --[12,25,43,76,123,131,211,321]
206
207
     mergeSort [423,13,31,225,76,21,1231,412,152,2,382,341,65,586,23]
208
209
     --expected
210
     --[2,13,21,23,31,65,76,152,225,341,382,412,423,586,1231]
211
212
     --6. (only for Haskell).
213
     --a. Define a function that takes two lists list1 and list2 and returns true if list1s elements are in list2.
214
215
     containElements [1,2,5,4,3,4,5,8,7] [1,2,3,4,5,6,7,8,9,0]
216
217
     --expected
     -- True
219
220
     containElements [9,9,4,6,6,6,11] [1,2,3,4,5,6,7,8,9,0]
221
222
     --expected
223
     --False
224
225
     containElements [22,34,5,4,3,2,1] [1,2,3,4,5,6,7,8,9,0,34,22]
226
```

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

3 C code

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

```
n=10;
             printf("for n=\%i are all pythagorean triples:\r\n", n);
25
             is_pyth(n, llist);
26
             display_list_array(llist. 3):
27
             llist = (struct NODE *)malloc(sizeof(struct NODE));
             n = 20:
             printf("for n=\%i are all pythagorean triples:\r\n", n);
             is_pyth(n, llist);
31
             display_list_array(llist, 3);
32
             llist = (struct NODE *)malloc(sizeof(struct NODE));
33
             n = 30:
34
             printf("for n=\%i are all pythagorean triples:\r\n", n);
35
             is_pyth(n, llist);
             display_list_array(llist, 3);
37
             llist = (struct NODE *)malloc(sizeof(struct NODE));
38
39
             printf("for n=\%i are all pythagorean triples:\r\n", n);
40
             is_pyth(n, llist);
             display_list_array(llist, 3);
             llist = (struct NODE *)malloc(sizeof(struct NODE));
             n = 50;
             printf("for n=\%i are all pythagorean triples:\r\n", n);
45
             is_pyth(n, llist);
46
             display_list_array(llist, 3);
47
48
             llist = (struct NODE *)malloc(sizeof(struct NODE));
             printf("3. A positive integer is perfect if it equals the sum of its factors, excluding the number itself. Define the function "
                      "that returns the list of all perfect numbers up to the given input number\r\n");
51
             n = 30;
52
             perfectInt(n, llist);
53
             printf("for n=\%i are perfect numbers:\r\n", n);
             display_list(llist);
             llist = (struct NODE *)malloc(sizeof(struct NODE));
             n=100;
             perfectInt(n, llist);
58
             printf("for n=\%i are perfect numbers:\r\n", n);
59
             display_list(llist);
60
             Ilist = (struct NODE *)malloc(sizeof(struct NODE));
61
             n = 500;
62
             perfectInt(n, llist);
63
             printf("for n=\%i are perfect numbers:\r\n", n);
64
             display_list(llist);
65
             llist = (struct NODE *)malloc(sizeof(struct NODE));
66
             n=1000:
             perfectInt(n, llist);
             printf("for n=\%i are perfect numbers:\r\n", n);
             display_list(llist);
             Ilist = (struct NODE *)malloc(sizeof(struct NODE));
71
             n=10000;
72
             perfectInt(n, llist);
73
             printf("for n=\%i are perfect numbers:\r\n", n);
74
             display_list(llist);
75
             struct TUPLE * tuple;
             printf("4. Define a function that takes a string containing digits and any other characters, and returns a pair where the first "
78
                      "component is the list of digits in the string in the order they occur in the input string, and the second pair contains the "
79
                                                                 9
```

printf("for n=%i is sum of squares: $\%i\r\n$ ", n, sum_square(n));

printf("2. Define the function that returns the list of all pythagorean triples"

"whose components greater than 0 and less that the given integer.\r\n");

llist = (struct NODE *)malloc(sizeof(struct NODE));

18 19

20

21

22

struct NODE *Ilist;

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

141

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

```
204
     /* this function expects as input three ints and output is also int
205
      * function controls if input values
206
      * are sides of pythagorean triangle
207
      * */
208
     int test_pyth(int a, int b, int c) {
              if (a*a + b*b == c*c)
                       return 1;
211
              else
212
                       return 0:
213
214
     /* this function expects as an input int and a linked list of NODEs;
215
      * firstly function get list of divisors, then sum them and tests if
      * the result is equal to actual number, if yes actual number is added
217
218
     void perfectInt(int n, struct NODE *Ilist) {
219
              struct NODE * Ilist1;
220
              int i:
221
              llist1 = (struct NODE *)malloc(sizeof(struct NODE));
222
              for (i = 1; i \le n; i++)
                       divisors(i, llist1);
224
                       if (sum(llist1)==i) {
225
                                 append_node(llist, i);
226
227
                       llist1 = (struct NODE *)malloc(sizeof(struct NODE));
              free(llist1);
231
     /* this function expects as an input int and a linked list of NODEs:
232
       * function fills input linked list with divisors of input integer n
233
      * except for number itself
234
      * */
235
     void divisors(int n, struct NODE *llist1) {
237
              for (i = 1; i < n; i++) {
238
                       if ((n \% i) == 0)
239
                                 append_node(llist1, i);
240
              }
241
     /* this function expects as an input array fo characters and a TUPLE;
243
      * function fills in cycle first part of input TUPLE with integers from input
244
      * char array and second part fills with rest characters
245
246
     void splitString(char* s, struct TUPLE *tuple) {
247
              int i;
248
              int length = strlen(s);
249
              for (i = 0; i < length; i++) {
250
                       if (s[i]>=48 \&\& s[i]<=57)
251
                                 add_number_to_tuple(tuple, s[i]);
252
                       else
253
                                 add_character_to_tuple(tuple, s[i]);
254
      /stthis function expexts as an input array of ints and one integer
257
      * (length of that array)
258
       * this function firstly recursively divide input list
259
      * in halves and then these peices are joined and ordered by function merge
260
261
     int* merge_sort(int* list, int length) {
              if (length \leq 1)
263
                       return list;
264
              int * left;
265
```

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

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

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