Haskell Hwk2

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

February 21, 2011

1 Haskell code

```haskell
module Hwk2 where
import Data.Char

-- 1.
-- 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

-- 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
-- is counted recursively
sumSquare :: Int -> Int
sumSquare 0 = 0
sumSquare n = n * n + sumSquare (n-1)

-- 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
-- and then the sum is counted by the function sum' declared before
sumSquare' :: Int -> Int
sumSquare' n = sum'[x * x | x <- [1..n]]

-- 2.
-- this function expects as an input triple of positive ints
-- and output is Bool value;function controls if members of given triple
-- are sides of pythagorean triangle
isPyth :: (Int, Int, Int) -> Bool
isPyth (a, b, c) = c * c == a * a + b * b

-- 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 if given sides are sides
-- of pythagorean triangle; for more efficiency it is also controlled if sides
-- ordinates a and b are lower than hypotenuse c and if this sides can create
-- 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,
    (a + b > c),isPyth (a, b, c)]

-- 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
-- ordinates a and b are lower than hypotenuse c and if this sides can create
-- 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,
    (a + b > c), isPyth' (a, b, c)]
    where isPyth' (a, b, c) = c * c == a * a + b * b

-- 3.
-- this function expects as an input list of ints and int and output is a list of ints;
-- function computes the 'mirror' divisors to a number when are divisors to root known
computeRestDivisors :: [Integer] -> Integer -> [Integer]
computeRestDivisors [] n = []
computeRestDivisors (x:xs) n = if ((x /= (div n x)) && (x/=1))
                                then x :((div n x) : (computeRestDivisors xs n))
                               else if (n/=x)
                                then x : (computeRestDivisors xs n)
                               else computeRestDivisors xs n

-- this function expects as an input int and output is a list of ints;
-- this function counts by the list generator divisors of given number
-- to the root of the number and number itself is omitted
findDivisors :: Integer -> [Integer]
findDivisors num = let root = round ( sqrt (fromInteger num)) in
                computeRestDivisors [number | number <- [1..root], mod num number == 0] num

-- this function expects as an input int and output is a list of ints;
-- function generate list and adds concrete number if his sum of divisors
-- equals to number itself (uses first implementation of 'findDivisors')
perfectInt :: Integer -> [Integer]
perfectInt n = [x |x <- [1..n], sum'(findDivisors x)==x]

-- this function expects as an input int and output is a list of ints;
-- this function counts by the list generator divisors of given number
-- to the number and number itself is omitted; this is expected to be
-- a slower implementation than function 'findDivisors'
findDivisors' :: Int -> [Int]
findDivisors' n = [x |x <- [1..(n-1)], mod n x == 0]

-- this function expects as an input int and output is a list of ints;
-- function generate list and adds concrete number if his sum of divisors
-- equals to number itself; it is similar function like function 'perfectInt'
-- but for control uses second implementation of 'findDivisors''
perfectInt' :: Int -> [Int]
perfectInt' n = [x |x <- [1..n], sum'(findDivisors' x)==x]

-- 4.
-- this function expects as an input a list of characters and output is a tuple
-- where first member are numbers extracted form the list
-- 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)
-- of the tuple; when is resulted tuple constructed, it is only taken into account
-- the member of the tuple according to his actual position (first or second)
splitString :: [Char] -> ([Char],[Char])
splitString [] = ([], [])
splitString (x:xs) = if isNumber x
                       then (x: (fst (splitString xs)), snd (splitString xs))
                       else (fst (splitString xs), x: (snd (splitString xs)) )
```

```
-- 5.
-- this function expects as an input a two lists of 'orderable' symbols
-- and output is ordered list of this symbols; the Ord class is used for
-- 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
-- two ordered lists into ordered list.
merge :: Ord a => [a] -> [a] -> [a]
merge [] [] = []
merge [] (x:[]) = [x]
merge (x:[]) [] = [x]
merge (x:[]) (y:[]) = if x <= y
                        then [x] ++ [y]
                      else
                        [y] ++ [x]
merge (x:xs) (y:[]) = if x <= y
                        then x : merge xs [y]
                      else
                        y : (x:xs)
merge (x:[]) (y:ys) = if y <= x
                        then y : merge ys [x]
                      else
                        x : (y:ys)

merge (x:xs) (y:ys) = if x <= y
                        then x : merge xs (y:ys)
                      else
                        y : merge (x:xs) ys

-- 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
-- in halves and then these peices are joined and ordered by function merge
mergeSort :: Ord a => [a] -> [a]
mergeSort [] = []
mergeSort (x:[]) = [x]
mergeSort xs = merge (mergeSort (fst (splitAt (div (length xs) 2) xs)))
                 (mergeSort (snd (splitAt (div (length xs) 2) xs)))

-- 6.
--a.
-- this function expects as an input orderable symbol and list of 'orderable' symbols
-- and output is boolean value; this function only tests, if given list contains
-- input symbol
containList :: Ord a => a -> [a] -> Bool
containList x [] = False
containList x (y:ys) = if x == y
                        then True
                       else containList x ys

-- 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
-- are contained in the second list; it takes one member after another and tests it;
-- when find first not contained member, then return False, otherwise when he reaches
-- empty first list, then returns True
containElements :: Ord a => [a] -> [a] -> Bool
containElements [] ys = True
containElements (x:xs) ys = if containList x ys
                              then containElements xs ys
                             else
```

```
                                  False

--b.
-- this function expects as an input two lists of 'orderable' symbols
-- 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
-- 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
subsequence [] [] = True
subsequence (x:xs) [] = False
subsequence (x:xs) (y:ys) = if x == y
                               then subsequence xs ys
                               else subsequence (x:xs) ys
```

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

sumSquare 13
-- expected 819

sumSquare 15
-- expected 1240

-- another option of sumSquare

sumSquare' 5
-- expected 55

sumSquare' 8
-- expected 204

sumSquare' 10
-- expected 385

sumSquare' 13
-- expected 819

sumSquare' 15
-- expected 1240

-- 2. Define the function that returns the list of all pythagorean triples
-- whose components greater than 0 and less that the given integer.

pyths 10

--expected
--[(3,4,5),(4,3,5),(6,8,10),(8,6,10)]
```

```

pyths 20

--expected
--[(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)]

pyths 30

--expected
--[(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),(15,8,17),(15,20,25),
--(16,12,20),(18,24,30),(20,15,25),(20,21,29),(21,20,29),(24,7,25),
--(24,10,26),(24,18,30)]

pyths 40

--expected
--[(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),
--(15,36,39),(16,12,20),(16,30,34),(18,24,30),(20,15,25),(20,21,29),(21,20,29),
--(21,28,35),(24,7,25),(24,10,26),(24,18,30),(24,32,40),(28,21,35),(30,16,34),
--(32,24,40),(35,12,37),(36,15,39)]

pyths 50

--expected
--[(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),
--(20,21,29),(21,20,29),(21,28,35),(24,7,25),(24,10,26),(24,18,30),(24,32,40),
--(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)]

-- another option of pyths

pyths' 10

--expected
--[(3,4,5),(4,3,5),(6,8,10),(8,6,10)]

pyths' 20

--expected
--[(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)]

pyths' 30

--expected
--[(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),(15,8,17),(15,20,25),
--(16,12,20),(18,24,30),(20,15,25),(20,21,29),(21,20,29),(24,7,25),
--(24,10,26),(24,18,30)]

pyths' 40

--expected
--[(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),
--(15,36,39),(16,12,20),(16,30,34),(18,24,30),(20,15,25),(20,21,29),(21,20,29),
--(21,28,35),(24,7,25),(24,10,26),(24,18,30),(24,32,40),(28,21,35),(30,16,34),
--(32,24,40),(35,12,37),(36,15,39)]

pyths' 50

--expected
--[(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),
--(20,21,29),(21,20,29),(21,28,35),(24,7,25),(24,10,26),(24,18,30),(24,32,40),
--(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)]

--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

perfectInt 30

--expected
--[6,28]

perfectInt 100

--expected
--[6,28]

perfectInt 500

--expected
--[6,28,496]

perfectInt 1000

--expected
--[6,28,496]

perfectInt 10000

--expected
--[6,28,496,8128]

-- another option of perfectInt

perfectInt' 30

--expected
--[6,28]

perfectInt' 100

--expected
--[6,28]

perfectInt' 500

--expected
--[6,28,496]

perfectInt' 1000

--expected
--[6,28,496]
```

```
perfectInt' 10000

--expected
--[6,28,496,8128]


--4. Define a function that takes a string containing digits and any other characters, and returns a pair
-- where the first component is the list of digits in the string in the order they occur in the input string,
-- and the second pair contains the rest of characters of the string in the order they occur.

splitString "Value#1 value#2 35 !"

--expected
--("1235","Value# value# !")

splitString "GreAT &&2*30!0 FuN B9o0Y!23#"

--expected
--("23009023","GreAT &&*! FuN BoY!#")

splitString "I23& 1jU1**s12T CA#32me T98#2O s#2Ay# He43LL2o!#?#"

--expected
--("231112329822432","I& jU**sT CA#me T#O s#Ay# HeLLo!#?#")

splitString "T56h44e65 O656nL4y4 W65A245y32 T32o32 HA23vE 2a34 FR#i$En$d I5S3 t[$$]3452O B76e 3A11 FRiE_32nD!(23)"

--expected
--("56446565644652453232322323453345276311322 3","The OnLy WAy To HAvE a FR#i$En$d IS t[$$]O Be A FRiE_nD!()")

--5. Define merge sort.

mergeSort [2,12,213,45,786,221,13,56,67,64,203]

--expected
--[2,12,13,45,56,64,67,203,213,221,786]

mergeSort [43,123,321,25,76,211,131,12]

--expected
--[12,25,43,76,123,131,211,321]

mergeSort [423,13,31,225,76,21,1231,412,152,2,382,341,65,586,23]

--expected
--[2,13,21,23,31,65,76,152,225,341,382,412,423,586,1231]

--6. (only for Haskell).
--a. Define a function that takes two lists list1 and list2 and returns true if list1s elements are in list2.

containElements [1,2,5,4,3,4,5,8,7] [1,2,3,4,5,6,7,8,9,0]

--expected
--True

containElements [9,9,4,6,6,6,11] [1,2,3,4,5,6,7,8,9,0]

--expected
--False

containElements [22,34,5,4,3,2,1] [1,2,3,4,5,6,7,8,9,0,34,22]

```

```
--expected
--True

containElements [2,5,7,9] [1,2,3,4,5,6,7,8,9,0]

--expected
--True

--b. Define a function that takes two lists list1 and list2 and returns true if list1 appears in list2 as a subsequence. To
--appear as a subsequence means that the elements in list1 appear in the same sequence in list2.

subsequence [22,34,5,4,3,2,1] [1,2,3,4,5,6,7,8,9,0,34,22]

--expected
--False

subsequence [2,5,7,9] [1,2,3,4,5,6,7,8,9,0]

--expected
--True

subsequence [12,15,17,22,35,71] [12,13,14,15,16,17,18,19,20,21,22,33,34,35,76,77,78]

--expected
--False

subsequence [12,15,17,22,35,77] [12,13,14,15,16,17,18,19,20,21,22,33,34,35,76,77,78]

--expected
--True

subsequence [1,5,7,9,12] [1,3,4,5,7,8,9,11,12,13]

--expected
--True

subsequence [1,5,7,9,12] [1,10,9,5,7,8,11,12,13]

--expected
--False
```

3 C code

```
#include <stdio.h>
#include <string.h>
#include "headers/node.h"
#include "headers/tuple.h"
#include "headers/functions.h"

int main() {
        printf("1. Define a function that computes the sum of the of the squares 1..n, given n\r\n");
        int n = 5;
        printf("for n=%i is sum of squares: %i\r\n", n, sum_square(n));
        n = 8;
        printf("for n=%i is sum of squares: %i\r\n", n, sum_square(n));
        n = 10;
        printf("for n=%i is sum of squares: %i\r\n", n, sum_square(n));
        n = 13;
        printf("for n=%i is sum of squares: %i\r\n", n, sum_square(n));
        n = 15;
```

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

    struct NODE *llist;
    llist = (struct NODE *)malloc(sizeof(struct NODE));
    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");
    n=10;
    printf("for n=%i are all pythagorean triples:\r\n", n);
    is_pyth(n, llist);
    display_list_array(llist, 3);
    llist = (struct NODE *)malloc(sizeof(struct NODE));
    n=20;
    printf("for n=%i are all pythagorean triples:\r\n", n);
    is_pyth(n, llist);
    display_list_array(llist, 3);
    llist = (struct NODE *)malloc(sizeof(struct NODE));
    n=30;
    printf("for n=%i are all pythagorean triples:\r\n", n);
    is_pyth(n, llist);
    display_list_array(llist, 3);
    llist = (struct NODE *)malloc(sizeof(struct NODE));
    n=40;
    printf("for n=%i are all pythagorean triples:\r\n", n);
    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);
    is_pyth(n, llist);
    display_list_array(llist, 3);

    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");
    n=30;
    perfectInt(n, llist);
    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);
    printf("for n=%i are perfect numbers:\r\n", n);
    display_list(llist);
    llist = (struct NODE *)malloc(sizeof(struct NODE));
    n=500;
    perfectInt(n, llist);
    printf("for n=%i are perfect numbers:\r\n", n);
    display_list(llist);
    llist = (struct NODE *)malloc(sizeof(struct NODE));
    n=1000;
    perfectInt(n, llist);
    printf("for n=%i are perfect numbers:\r\n", n);
    display_list(llist);
    llist = (struct NODE *)malloc(sizeof(struct NODE));
    n=10000;
    perfectInt(n, llist);
    printf("for n=%i are perfect numbers:\r\n", n);
    display_list(llist);

    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 "
           "component is the list of digits in the string in the order they occur in the input string, and the second pair contains the "
```

```
            "rest of characters of the string in the order they occur.\r\n");
    tuple = (struct TUPLE *)malloc(sizeof(struct TUPLE));
    char * str ="Value#1 value#2 35 !";
    init_tuple(tuple, str);
    splitString(str, tuple);
    printf("for \"%s\" is requested tuple:\r\n", str);
    print_tuple(tuple);
    tuple = (struct TUPLE *)malloc(sizeof(struct TUPLE));
    str ="GreAT &&2*30!0 FuN B9o0Y!23#";
    init_tuple(tuple, str);
    splitString(str, tuple);
    printf("for \"%s\" is requested tuple:\r\n", str);
    print_tuple(tuple);
    tuple = (struct TUPLE *)malloc(sizeof(struct TUPLE));
    str ="I23& 1jU1**s12T CA#32me T98#2O s#2Ay# He43LL2o!#?#";
    init_tuple(tuple, str);
    splitString(str, tuple);
    printf("for \"%s\" is requested tuple:\r\n", str);
    print_tuple(tuple);
    tuple = (struct TUPLE *)malloc(sizeof(struct TUPLE));
    str ="T56h44e65 O656nL4y4 W65A245y32 T32o32 HA23vE 2a34 FR#i$En$d I5S3 t[$$]3452O B76e 3A11 FRiE_32nD!(23)";
    init_tuple(tuple, str);
    splitString(str, tuple);
    printf("for \"%s\" is requested tuple:\r\n", str);
    print_tuple(tuple);

    printf("5. Define merge sort.\r\n");
    int length = 11;
    int * list = malloc(length*sizeof(int));
    list[0]=2;
    list[1]=12;
    list[2]=213;
    list[3]=45;
    list[4]=786;
    list[5]=221;
    list[6]=13;
    list[7]=56;
    list[8]=67;
    list[9]=64;
    list[10]=203;
    printf("For unsorted array:\r\n");
    array_output(list, length);
    int * sorted_list = merge_sort(list, length);
    printf("is sorted array:\r\n");
    array_output(sorted_list, length);
    length = 8;
    list = malloc(length*sizeof(int));
    list[0]=43;
    list[1]=123;
    list[2]=321;
    list[3]=25;
    list[4]=76;
    list[5]=211;
    list[6]=131;
    list[7]=12;
    printf("For unsorted array:\r\n");
    array_output(list, length);
    sorted_list = merge_sort(list, length);
    printf("is sorted array:\r\n");
    array_output(sorted_list, length);
    length = 15;
    list = malloc(length*sizeof(int));
    list[0]=423;
    list[1]=13;
    list[2]=31;
    list[3]=225;
    list[4]=76;
    list[5]=21;
    list[6]=1231;
    list[7]=412;
    list[8]=152;
    list[9]=2;
    list[10]=382;
    list[11]=341;
    list[12]=65;
    list[13]=586;
    list[14]=23;
    printf("For unsorted array:\r\n");
    array_output(list, length);
    sorted_list = merge_sort(list, length);
    printf("is sorted array:\r\n");
    array_output(sorted_list, length);

    free(llist);
    free(tuple);
    free(list);
    free(sorted_list);

    return 0;
}
/* input of this function is int ad this function also returns int
 * in the for cycle counts square of number 1 to n and add the result to sum
 * */
int sum_square(int n) {
    int i;
    int sum = 0;
    for (i = 1; i <= n; i++)
        sum+=i*i;
    return sum;
}
/* this function expects as an input int and NODE pointer
 * and output is a list of triples; in this function are three sides of triangle
 * controlled by the function test_pyth if given sides are sides
 * of pythagorean triangle; if yes they are add to the linked list
 * of NODEs in array
 * */
void is_pyth(int n, struct NODE *llist) {
    int* result;
    int c;
    int a;
    int b;
    for (c = 1; c <= n; c++) {
        for (a = 1; a < c; a++) {
            for (b = 1; b < c; b++) {
                if (test_pyth(a, b, c)) {
                    result = malloc(3 * sizeof(int));
                    result[0] = a;
                    result[1] = b;
                    result[2] = c;
                    append_node_array(llist, result);
                }
            }
        }
    }
```

```
}
/* this function expects as input three ints and output is also int
 * function controls if input values
 * are sides of pythagorean triangle
 * */
int test_pyth(int a, int b, int c) {
        if (a*a + b*b == c*c)
                return 1;
        else
                return 0;
}
/* this function expects as an input int and a linked list of NODEs;
 * 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
 * */
void perfectInt(int n, struct NODE *llist) {
        struct NODE * llist1;
        int i;
        llist1 = (struct NODE *)malloc(sizeof(struct NODE));
        for (i = 1; i <= n; i++) {
                divisors(i, llist1);
                if (sum(llist1)==i) {
                        append_node(llist, i);
                }
                llist1 = (struct NODE *)malloc(sizeof(struct NODE));
        }
        free(llist1);
}
/* this function expects as an input int and a linked list of NODEs;
 * function fills input linked list with divisors of input integer n
 * except for number itself
 * */
void divisors(int n, struct NODE *llist1) {
        int i;
        for (i = 1; i < n; i++) {
                if ((n % i) == 0)
                        append_node(llist1, i);
        }
}
/* this function expects as an input array fo characters and a TUPLE;
 * function fills in cycle first part of input TUPLE with integers from input
 * char array and second part fills with rest characters
 * */
void splitString(char* s, struct TUPLE *tuple) {
        int i;
        int length = strlen(s);
        for (i = 0; i< length; i++) {
                if (s[i]>=48 && s[i]<=57)
                        add_number_to_tuple(tuple, s[i]);
                else
                        add_character_to_tuple(tuple, s[i]);
        }
}
/*this function expexts as an input array of ints and one integer
 * (length of that array)
 * this function firstly recursively divide input list
 * in halves and then these peices are joined and ordered by function merge
 * */
int* merge_sort(int* list, int length) {
        if (length <= 1)
                return list;
        int * left;
```

```
    int * right;
    int * result;
    int middle = length / 2;
    left = malloc(middle*sizeof(int));
    right = malloc((length-middle)*sizeof(int));
    result = malloc(length*sizeof(int));
    int i;
    for (i = 0; i < middle; i++)
        left[i] = list[i];
    for (i = middle; i < length; i++)
        right[i-middle] = list[i];

    left = merge_sort(left, middle);
    right = merge_sort(right, (length -middle));
    result = merge(left, right, middle, (length -middle));
    free(left);
    free(right);
    return result;
}
/* this function expexts as an input two arrays of ints and two integers
 * (lengths of that arrays)
 * this function is used for joining two ordered lists into ordered list
 * */
int* merge(int* left, int* right, int lengthLeft, int lengthRight) {
    int* result = malloc((lengthLeft+lengthRight)*sizeof(int));
    int resultPointer=0;
    int leftPointer=0;
    int rightPointer=0;
    while (lengthLeft > leftPointer || lengthRight > rightPointer) {
        if (lengthLeft > leftPointer && lengthRight > rightPointer) {
            if (left[leftPointer] <= right[rightPointer]) {
                result[resultPointer] = left[leftPointer];
                resultPointer++;
                leftPointer++;
            } else {
                result[resultPointer] = right[rightPointer];
                resultPointer++;
                rightPointer++;
            }
        } else if (lengthLeft > leftPointer) {
            result[resultPointer] = left[leftPointer];
            resultPointer++;
            leftPointer++;
        } else if (lengthRight > rightPointer) {
            result[resultPointer] = right[rightPointer];
            resultPointer++;
            rightPointer++;
        }
    }
    return result;

}
void array_output(int* list, int length) {
    int i;
    for (i = 0; i < length; i++)
        printf("%i ", list[i]);
    printf("\r\n");
}
```

```
#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 ", llist->number);
                llist = llist->next;
        }
        printf("%d \r\n", llist->number);
}
void display_list_array(struct NODE *llist, int length) {
        if (llist->next != NULL)
                llist = llist->next;
        int i;
        while (llist->next != NULL) {

                for (i = 0; i < length; i++) {
                        printf("%i ", llist->array[i]);
                }
                printf("\r\n");
                llist = llist->next;
        }
        if (llist->array != NULL) {
                for (i = 0; i < length; i++) {
                        printf("%i ", llist->array[i]);
                }
                printf("\r\n");
        }
}
void append_node(struct NODE *llist, int num) {
        while (llist->next != NULL)
                llist = llist->next;

        llist->next = (struct NODE *)malloc(sizeof(struct NODE));
        llist->next->number = num;
        llist->next->next = NULL;
}
void append_node_array(struct NODE *llist, int* array) {
        while (llist->next != NULL)
                llist = llist->next;

        llist->next = (struct NODE *)malloc(sizeof(struct NODE));
        llist->next->array = array;
        llist->next->next = NULL;
}
int sum(struct NODE *llist) {
        int sum;
        if (llist->next != NULL)
                llist = llist->next;
        while (llist->next != NULL) {
                sum += llist->number;
                llist = llist->next;
        }
        sum += llist->number;
        return sum;
}
```

```c
#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;
        tuple->pointerCharacters = 0;
}
void add_number_to_tuple(struct TUPLE *tuple, char num) {
        tuple->numbers[tuple->pointerNumbers] = num;
        tuple->pointerNumbers++;
}
void add_character_to_tuple(struct TUPLE *tuple, char c) {
        tuple->characters[tuple->pointerCharacters] = c;
        tuple->pointerCharacters++;
}
void print_tuple(struct TUPLE *tuple) {
        printf("(\"%s\",\"%s\")", tuple->numbers, tuple->characters);
        printf("\r\n");
}
```