Exercise 10/2

(Question 1)

By

Sanaz Rahbari

Introduction to programming C

Lecturer

Prof.-Dr. Ulrich Hauser-Ehninger

Ilia State University

Tbilisi

2019

/* The below program is calculated square of a polynomial and first, it prints the polynomial itself and on the next line, its square. */

```
1 # include <stdio.h>
2 # include <stdlib.h>
3
```

/* The program has written in the way that through lines 4-8, it defines typedef structure to indicate parameters of the polynomial. Coefficient is an array of float numbers which is multiplying the variable. But as it does not have specific size, now, it is better to be mention as a pointer. The size of polynomial describes the number of expressions. */

```
4 typedef struct s_polynomial
5  {
6   float * coefficients ;
7   unsigned int size ;
8  } Polynomial ;
9
```

/* In line 10 a function has created to initiate the polynomial and has pointer to polynomial and an unsigned int for size as its parameters. This function is void type which it will not return anything at the end. In in line 13-16 said that if the number of expression is not zero, free the space that the program got from heap in memory for polynomial coefficient; else, line 17-18, it must dedicate space from heap to the polynomial coefficient in size of float for "n" numbers.
The reason of using "malloc" here is that as at the beginning we are not aware of the exact length of the float coefficient as this parameter is an array, so there is a need of coefficient, thus, at the beginning in the polynomial structure it has been mention by pointer and for the same reason size in line 17 mention by pointer. */

```
10  void initPoly ( Polynomial * polynomial , unsigned int size )
11  {
12 unsigned int index ;
13 if (0 != polynomial -> size )
14   {
15  free ( polynomial -> coefficients );
16   }
17 polynomial -> coefficients = ( float *) malloc ( sizeof ( float ) * size );
18 polynomial -> size = size ;
19 for ( index = 0; polynomial -> size > index ; ++ index )
20  {
21 polynomial -> coefficients [ index ] = 0;
22  }
```

```
23 }
24
```
/* From lines 25-32, another function created to finish the polynomial and it said that if the number of size is not zero free the polynomial coefficient and exit and if it is not give polynomial size zero and then exit. Also as this function is again void it is not return anything.

```
25 void exitPoly ( Polynomial * polynomial )
26 {
27 if (0 != polynomial -> size )
28 {
29 free ( polynomial -> coefficients );
30 }
31 polynomial -> size = 0;
32 }
33
34 void polySquare ( Polynomial polynomial , Polynomial * result )
35 {
36 unsigned int indexFirst ;
37 unsigned int indexSecond ;
38 unsigned int sizeResult = ( polynomial . size - 1) * 2 + 1;
39
```
/* In lines 40-50, another void function delared with result and sizeResult as its parameters. Here the result of square of polynomial will be calculated. In order to do that. Two for loops has mentioned. The first loop is for the first index and the second one for second index which both start from index zero. And then the result of multiplication of coefficient first and and second index will add to the coefficient result at each duration of loop. */

```
40 initPoly ( result , sizeResult );
41 for ( indexFirst = 0; polynomial . size > indexFirst ;++ indexFirst )
42  {
43 for ( indexSecond = 0; polynomial . size >indexSecond ; ++ indexSecond )
44  {
45 result -> coefficients [ indexFirst + indexSecond ] += polynomial . coefficients [ indexFirst]
46  * polynomial . coefficients [ indexSecond ];
47
48   }
49  }
50 }
51
52 void printPolynomial ( Polynomial polynomial )
```

```
53 {
54 unsigned int index ;
55 for ( index = 0; polynomial . size > index ; ++ index )
56{
/* Lines 57-64, is about printing of polynomial, line 61 said that fist print float coefficient
next to the algebra variable and then the integer exponent for each index and if the index
was not zero add “+” between each expression (line 59). Then by mentioning line 63
everything that has been saved in this loop will be printed in console. */

57 if (0 != index )
58 {
59 printf ("+");
60}
61 printf ("%fx ^%d", polynomial . coefficients [ index ], index );
62}
63 printf ("\n");
64 }
65
66 int main ( void )
67 {
/* In lines 69 and 71, parameters polynomial size and polynomial result are initialize to be
started from zero in main function.
In main function some parameters is called by address to go to the location address and put
command below in that address but some of them delared by pointer to, which are in preintf
function to get content of those parameters.  */

68 Polynomial polynomial ;
69 polynomial . size = 0;
70 Polynomial result ;
71 result . size = 0;
72
73 initPoly (& polynomial , 2);
73 polynomial . coefficients [0] = 2;
75 polynomial . coefficients [1] = 3;
76 printf (" Polynomial :\t");
77 printPolynomial ( polynomial );
78 polySquare ( polynomial , & result );
79 printf (" Squared :\t");
80 printPolynomial ( result );
81
82 initPoly (& polynomial , 4);
```

```
83 polynomial . coefficients [0] = 1;
84 polynomial . coefficients [1] = 2;
85 polynomial . coefficients [2] = -3;
86 polynomial . coefficients [3] = 4;
87 printf (" Polynomial :\t");
88 printPolynomial ( polynomial );
89 polySquare ( polynomial , & result );
90 printf (" Squared :\t");
91 printPolynomial ( result );
92
93 exitPoly (& polynomial );
94 exitPoly (& result );
95 return 0 ;
96 }
```