Project Report for Programming

Group B2

- **Project name:** Number Extrapolation

- **Members of our group and their duties:**
  - Basso Nick - Technical Lead & Project Manager
  - Petranis Ann - Project Reviewer and Manager
  - Pascaru Stefan-Alin – Developer of Newton's Interpolation algorithm implementation
  - Chisnicean Ivan - Tester
  - Corman Daniel – Quality Assurance

- **The task of our project** is estimating the 101st number of the set given to us from A groups, using the extrapolation method. We were able to use 3 algorithms: linear, polynomial and conic. Linear extrapolation is to create a tangent line at the end of the known data and to extend it beyond that limit. Polynomial extrapolation is to use Lagrange interpolation or Newton's method of finite differences to create a Newton series that fits the data. Conic extrapolation could be done with a conic sections template, which can be created using five points near the end of the known data. Thus, we decided to implement all this methods to obtain the most exact result.

- **Algorithm**

  Our code begins with the declaration of libraries that are necessary for successful compilation and launch. So, we used `#include <stdio.h>` which stands for "standard input output" header file.

  Our program contains 5 functions for the number extrapolation:

```
void linearTwoPointsExtr();
void linearDeltaSumExtr();
void linearDeltaCoefExtr();
void conicSectionExtr();
void polynomLagrExtr();
```

These functions will be invoked next in main function of the program.

Of course, we used global variables `int numElem` for the number of elements which gave us A groups and `float arr[10000 + 5]` that represents a float-type array of 10000 + 5 elements for the input.

Our program begins with the data input. All necessary values are stored in the file input.txt.

```
// File input declarations
FILE *fp;
fp = fopen("input.txt", "r");

// entering data from the file
int i;

fscanf(fp, "%d", &numElem);

for(i = 0; i < numElem; i++){
       fscanf(fp, "%f", &arr[i]);
}
```

Next starts first function `linearTwoPointsExtr()` - linear extrapolation for two last numbers in the array. In this function we used variables: **y1** - value of the array element at a lower index (lower index = first point index), **longXDelta** - difference between index at which value to be extrapolated and first point index, **shortXDelta** - difference between index at second point and index at first point **yDelta** - difference between array values at first and second point indices and **result** - answer given basing on that function value dependency of argument is linear.

```
void linearTwoPointsExtr(){
       float y1 = arr[numElem - 2];
       float longXDelta = (numElem + 1) - (numElem - 1);
       float shortXDelta = numElem - (numElem - 1);
       float yDelta = arr[numElem - 1] - arr[numElem - 2];
       float result = y1 + longXDelta / shortXDelta * yDelta;

       printf("Linear extrapolation result:\t\t\t\t       %f\n", result);
}
```

After that `linearDeltaSumExtr()` function is induced. It fulfills linear delta sum extrapolation. For this function work next variables: **delta** - sum of all differences between two consecutive array elements, **averageDelta** - average of all differences between two consecutive array elements, **result** - averageDelta + last array element value.

```
void linearDeltaSumExtr(){
       int i;
       float delta = 0.0, averageDelta, result;
```

```
	for(i = 1; i < numElem; i++){
		delta += arr[i] - arr[i - 1];
	}

	averageDelta = delta / (numElem - 1);

	result = arr[numElem - 1] + averageDelta;

	printf("Linear delta sum extrapolation result:\t\t\t\t	%f\n", result);
}
```

Then program invokes the function of linear delta coefficient extrapolation (`linearDeltaCoefExtr()`). In this function are used variables: **delta** - sum of all coefficients that represent, division of current array element on the previous, **averageDelta** - average of all coefficients described in delta, **result** - last array element value multiplied by averageDelta.

```
void linearDeltaCoefExtr(){
	int i;
	float delta = 0.0, averageDelta, result;

	for(i = 1; i < numElem; i++){
		delta += arr[i] / arr[i - 1];
	}

	averageDelta = delta / (numElem - 1);

	result = arr[numElem - 1] * averageDelta;

	printf("Linear delta average multiplication cofficient extrapolation result:
	%f\n", result);
}
```

Next runs conic section extrapolation function `conicSectionExtr()`. In this function we used following variables: **xt** - stores the sum of all five last x coordinates (which is sum of all array element indices from 1 to number of elements inclusive in this specific task), **yt** - stores the sum of last five numbers given in the input, **xAvg** - average x-coordinate value obtained from the numbers in xt, **yAvg** - average y-coordinate value obtained from the numbers in yt (five last numbers in the input), **result** - multiplication of x and y coordinate averages multiplied by the number of points taken in examination.

```
void conicSectionExtr(){
	float result, xt = 0, yt = 0;
	float yAvg = 0.0, xAvg = 0.0;
	int i;

	for(i = numElem - 1; i >= numElem - 5; i--){
		xt += i + 1;
		yt += arr[i];
	}

	xAvg = (numElem + 1) / xt;
	yAvg = yt / 5;

	result = yAvg * xAvg * 5;

	printf("Conic section extrapolation result:\t\t\t\t	%f\n", result);
}
```

Finally, our program calls `polynomLagrExtr()` function, which means Lagrange polynomial extrapolation. For this function work next variables: **x** - point, value at which needs to be extrapolated, **k** - Lagrange coefficient obtained by multiplying results between all pairs of x-coordinates where one is obligatory not equal to the other, **result** - equals to the sum of all k-coefficients multiplied by the current y-coordinate value.

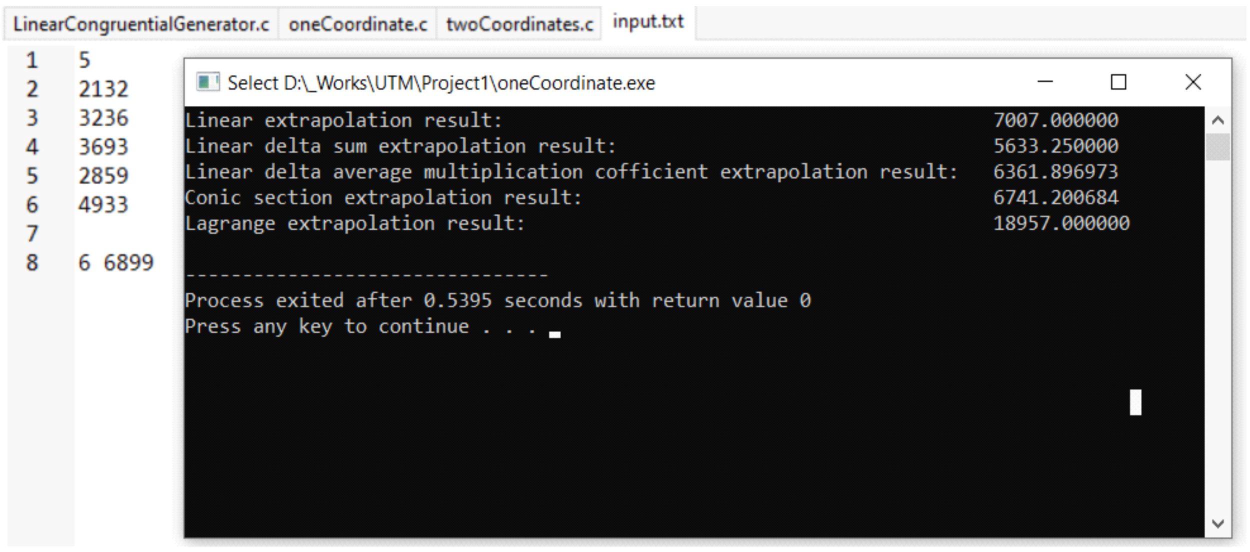
```
void polynomLagrExtr(){
//FILE *out1 = fopen("test1.txt", "w");
	float result = 0.0;
	float k;
	float x = numElem + 1;
	int i, j;

	for(j = 0; j < numElem; j++){
		k = 1;
		for(i = 0; i < numElem; i++){
			if(i != j){
				k *= (x - (i + 1)) / ((j + 1) - (i + 1));
				//fprintf(out1, "%d %d %f %f\n", i + 1, j + 1, x, k);
			}
		}
		result += arr[j] * k;
	}

	printf("Lagrange extrapolation result:\t\t\t\t\t	%f\n", result);
}
```

There is also a side task implementation of Newton's interpolation algorithm made by Stefan-Alin Pascaru.

```c
void polynomNewtIntr()
{
    float p[10000], f=arr[0], f1=1, f2=0;
    int k=numElem-1,i,j=1;

    do
    {
        for (i=0;i<numElem-1;i++)
        {
            p[i] = ((arr[i+1]-arr[i])/j);
            arr[i]=p[i];
        }
        f1=1;
        for(i=0;i<j;i++)
            {
                f1*=(k-i+1);
            }
        f2+=(arr[0]*f1);
        numElem--;
        j++;
    }
        while(numElem!=1);
    f+=f2;
    printf("Newton Polynomial result:\t\t\t\t\t       %f", f);
}
```

- **Results**

  After all these operations we get 5 results. Each result provided may differ from the others because of different extrapolation algorithms used and inputs generated in various ways. Next image will provide an example of the result which we have received after some tests.


LinearCongruentialGenerator.c
oneCoordinate.c
twoCoordinates.c
input.txt
Select D:_Works\UTM\Project1\oneCoordinate.exe

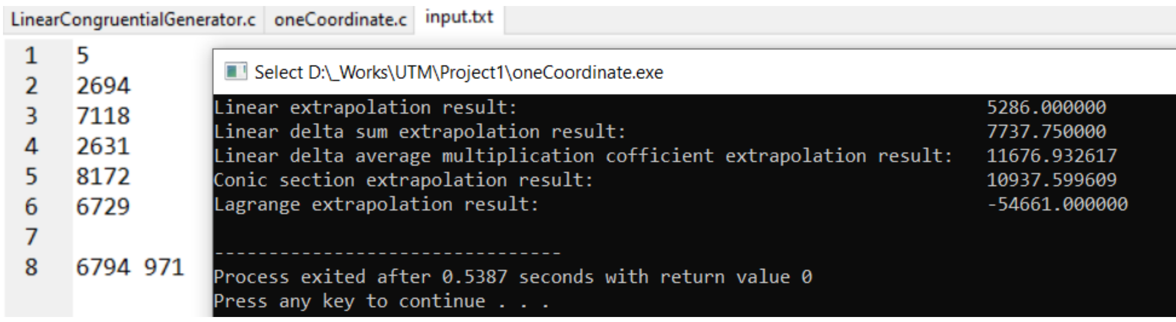
```
5
2132
3236
3693
2859
4933

6  6899
```

```
Linear extrapolation result:                                           7007.000000
Linear delta sum extrapolation result:                                 5633.250000
Linear delta average multiplication cofficient extrapolation result:   6361.896973
Conic section extrapolation result:                                    6741.200684
Lagrange extrapolation result:                                         18957.000000

--------------------------------
Process exited after 0.5395 seconds with return value 0
Press any key to continue . . .
```

- Analysis of results

In the process of solving the code problem, our team decided to do more tests. From our point of view, it is best to compare more results in order to understand exactly where and which procedures work better than others. As a result, we decided to do 5 tests that helped us in the analysis and the first one have been provided in previous section. Every test have 5 results, because of the 5 algorithms that are used in the Extrapolation method. In the TEST 1 there are 5 pseudo-random numbers that are given to us. The task is to find the next one in the most precisely way. We can see that the real NEXT NUMBER is 6899, but unfortunately we do not have such a number as a result. Nevertheless, we can find the most accurate result available. In this case it is number 7007.000000, which is a result of the linear extrapolation. The worst result is the result of the Lagrange extrapolation algorithm.

The next one is TEST 2. It is very similar to the first one, but the results are different. Next image will provide information about TEST 2.

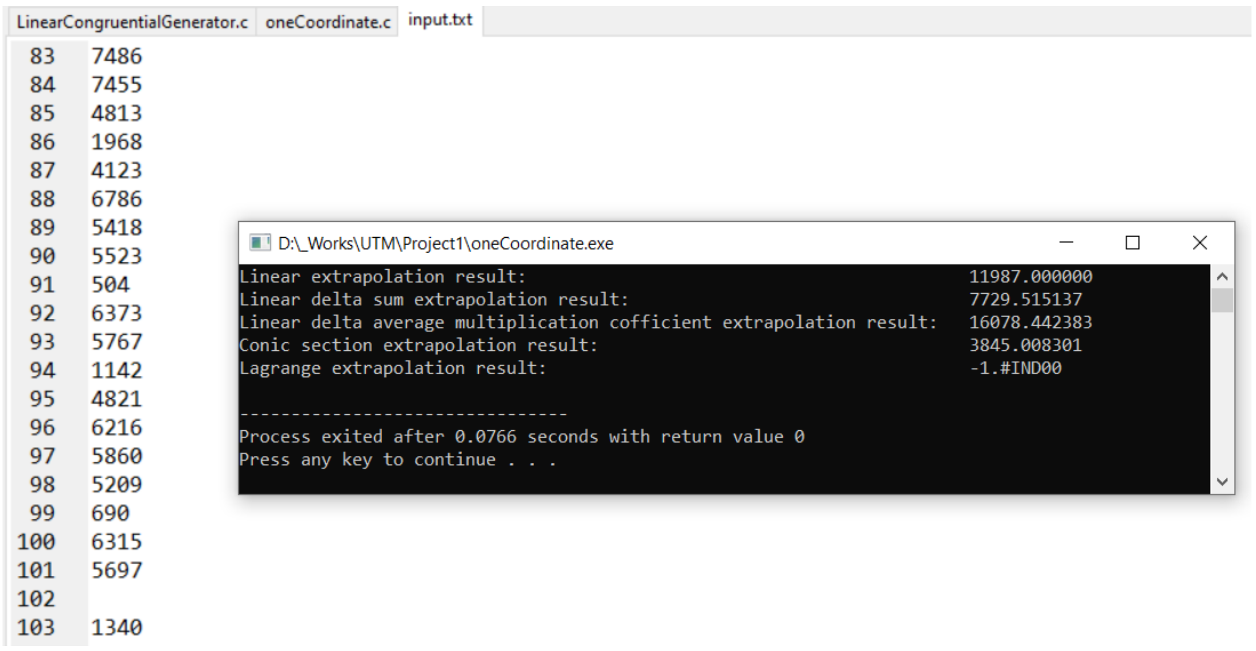

LinearCongruentialGenerator.c
oneCoordinate.c
input.txt
Select D:_Works\UTM\Project1\oneCoordinate.exe

```
5
2694
7118
2631
8172
6729

6794  971
```

```
Linear extrapolation result:                                           5286.000000
Linear delta sum extrapolation result:                                 7737.750000
Linear delta average multiplication cofficient extrapolation result:   11676.932617
Conic section extrapolation result:                                    10937.599609
Lagrange extrapolation result:                                         -54661.000000

--------------------------------
Process exited after 0.5387 seconds with return value 0
Press any key to continue . . .
```

In this case the best result is given by the Linear delta sum extrapolation algorithm, still a linear one. In other words, linear extrapolation and its variations work not bad for the small quantity of numbers, just like in the TEST 1 and TEST 2, while Lagrange extrapolation algorithm is the worst.

The TEST 3 is different from TEST 1 and 2, because it is a particular case, in which there are more than 5 pseudo-random numbers. The next image presents the information about TEST 3.

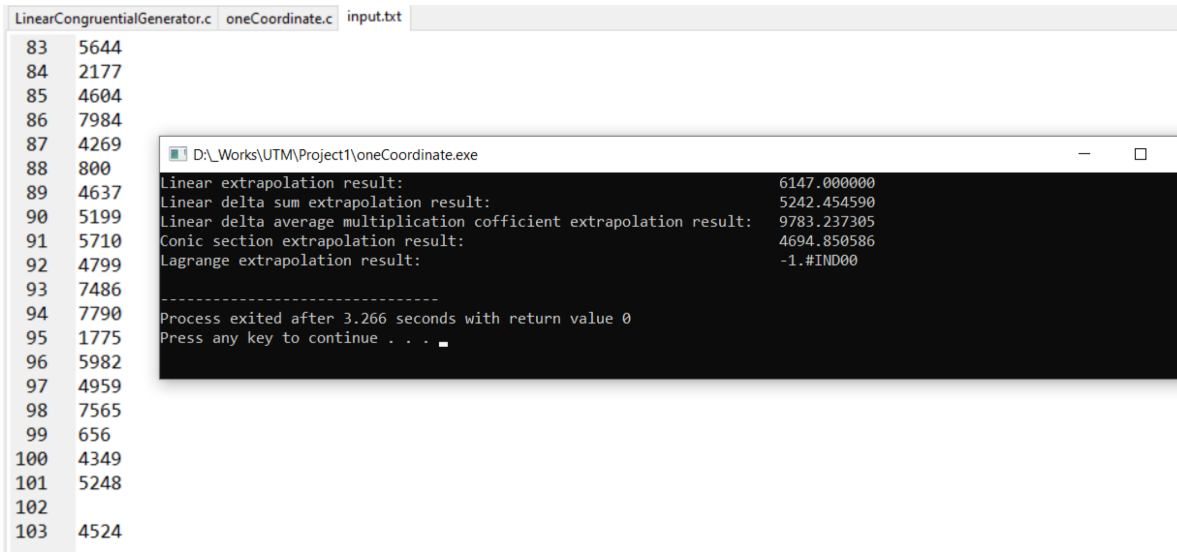

LinearCongruentialGenerator.c
oneCoordinate.c
input.txt
D:_Works\UTM\Project1\oneCoordinate.exe

```
7486
7455
4813
1968
4123
6786
5418
5523
504
6373
5767
1142
4821
6216
5860
5209
690
6315
5697

1340
```

```
Linear extrapolation result:                                11987.000000
Linear delta sum extrapolation result:                      7729.515137
Linear delta average multiplication cofficient extrapolation result:   16078.442383
Conic section extrapolation result:                         3845.008301
Lagrange extrapolation result:                              -1.#IND00

--------------------------------
Process exited after 0.0766 seconds with return value 0
Press any key to continue . . .
```

The first thing that we observe is the fact than Lagrange result is not at all the one that should be. However we can find the most accurate result. After some mathematic calculation we can say that Conic section extrapolation result is the best one. Still we need other test to proof or disproof that Conic extrapolation can find the most accurate answer for an example, which contains more numbers.

The TEST 4 is similar to the TEST 3. It looks like this:

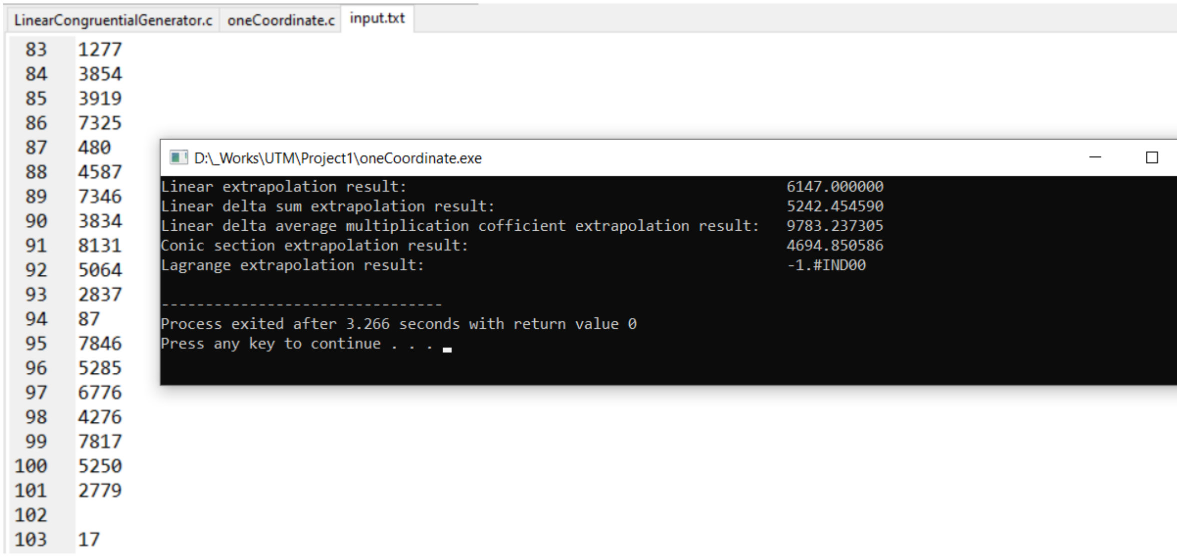


```
LinearCongruentialGenerator.c   oneCoordinate.c   input.txt
83   5644
84   2177
85   4604
86   7984
87   4269
88   800
89   4637
90   5199
91   5710
92   4799
93   7486
94   7790
95   1775
96   5982
97   4959
98   7565
99   656
100  4349
101  5248
102
103  4524
```

```
D:\_Works\UTM\Project1\oneCoordinate.exe
Linear extrapolation result:                                            6147.000000
Linear delta sum extrapolation result:                                  5242.454590
Linear delta average multiplication cofficient extrapolation result:    9783.237305
Conic section extrapolation result:                                     4694.850586
Lagrange extrapolation result:                                          -1.#IND00

--------------------------------
Process exited after 3.266 seconds with return value 0
Press any key to continue . . .
```

The situation is very interesting, because it is similar to the TEST 3. The best result is given by the Conic section extrapolation algorithm. The difference is only ~170 and that is amazing, because it cannot be accidental. The Lagrange extrapolation result are absolutely not the best one.

The last test, TEST 5 looks like previous two. The next image provides the results of the last test.



```
LinearCongruentialGenerator.c   oneCoordinate.c   input.txt
83   1277
84   3854
85   3919
86   7325
87   480
88   4587
89   7346
90   3834
91   8131
92   5064
93   2837
94   87
95   7846
96   5285
97   6776
98   4276
99   7817
100  5250
101  2779
102
103  17
```

```
D:\_Works\UTM\Project1\oneCoordinate.exe
Linear extrapolation result:                                            6147.000000
Linear delta sum extrapolation result:                                  5242.454590
Linear delta average multiplication cofficient extrapolation result:    9783.237305
Conic section extrapolation result:                                     4694.850586
Lagrange extrapolation result:                                          -1.#IND00

--------------------------------
Process exited after 3.266 seconds with return value 0
Press any key to continue . . .
```

TEST 5 is identical in the efficiency of the algorithms with TEST 3 and TEST 4. The Lagrange's result is the same in the last three examples, but the most important thing is the fact that the Conic section extrapolation gives us the best result, again.

In conclusion, our team decided to use the result of the Conic section extrapolation method as the most efficient way to find the next number of a pseudo-random set of numbers, because it gives the best results in the TEST 3, TEST 4 and TEST 5.