LAB#3 IEEE FLOATING POINT REPRESENTATION

CST8233 W2021

ALGONQUIN COLLEGE | to transform hopes and dreams into lifelong success

LAB OBJECTIVE

The objective of this lab is to get familiar with the following:

1- Overflow and Underflow in floating Point Representation

2- Absolute and Relative Error

Earning

To earn your mark for this lab, each student should finish the lab's requirements within the lab session and demonstrate the working code to the instructor.

STATEMENT OF THE PROBLEM:

Part A:

The following program demonstrate the overflow and underflow

Task A.1: Copy and paste the following code to your Visual Studio environment.

Task A.2: Run the code and notice the output of the program. Explain what happens to your instructor.

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

#include<math.h>
int main()
{
      int i;
      float n, x;
      n = 1.0;
      for (i = 0; i <= 127; i++)
      {
            n = n * 2.0;
            x = 1.0 / n;
                  printf("%d %e %E\n", i, x, n);
      }

return 0;
}
```

Task A.3: Change the loop's determinant "i" to few numbers higher than 127 and notice the output. Explain the results to your lab instructor.

Task A.4: replace the *floats* by *doubles* and notice the output. Explain the results to your lab instructor.

Part B:

Task B.1: Run the code and notice the output of the program. Explain what happens to your instructor.

```
#include <iostream>
#include <stdio.h>
using namespace std;

int main( )
{
  float x = 77777.0;
  float y = 7.0;
  float y1 = 1.0 / y;
  float z = x / y;
  float z1 = x * y1;

  if ( z != z1 ) {
    cout << z << " != " << z1 << endl;
    printf( "%1.20f != %1.20f\n", z, z1 );
  }
  else {
    cout << z << " == " << z1 << endl;
    printf( "%1.20f == %1.20f\n", z, z1 );
  }

  return 0;
}
```

Task B.2: replace the *floats* by *doubles* and notice the output. Explain the results to your lab instructor.

Task B.3: replace the *doubles by* floats , 77777.0 by 88888.0 and 7.0 by 8.0, then notice the output. Explain the results to your lab instructor.

Part C:

Task C.1: Run the code and notice the output of the program. Explain what happens to your instructor

```
#include <stdio.h>

int main()
{
	float f1 = 1.0;
	float f2 = 0.0;
	int i;
	for (i = 0; i < 10; i++)
		f2 += 1.0 / 10.0;
	printf("0x%08x 0x%08x\n", *(int*)&f1, *(int*)&f2);
	printf("f1 = %10.9f\n", f1);
	printf("f2 = %10.9f\n\n", f2);
	return 0;
}
```

Task C.2: replace 10 by 8   in the for loop and 10.0 by 8.0 in the loop body, notice the output. Explain the results to your lab instructor.

Part D:

Truncation error occurs when the numerical methods used for solving mathematical problem. The numerical method approximates your function using a finite number of terms.

The difference between the exact and approximate solution is called truncation error. The following task demonstrate the truncation error to compute the famous factorial function.

There is a famous numerical approximate formula, that gives an approximate evaluation for n!.

$$n! \cong \left(\frac{n}{e}\right)^n \sqrt{2\pi n}.$$

Write a C\C++ program to output a table with the following format, you may use the factorial function from Lab#1 to calculate the analytical value of factorial function at each step.

If your computer system does not have a predefined value for π,

Use either $\pi = \text{acos}(-1.0)$ OR $\pi = 4.0 * \text{atan}(1.0)$

Sample output:

Enter the number to calculate its factorial: 10

N	***n*!**	**approximation**	**Absolute error**	**Relative error**
1	**1**	**0.922137**	**0.077863**	**0.077863**
2	**2**	**1.919004**	**0.080996**	**0.040498**
3	**6**	**5.836210**	**0.163790**	**0.027298**
4	**24**	**23.506175**	**0.493825**	**0.020576**
5	**120**	**118.019168**	**1.980832**	**0.016507**
6	**720**	**710.078185**	**9.921815**	**0.013780**
7	**5040**	**4980.395832**	**59.604168**	**0.011826**
8	**40320**	**39902.395453**	**417.604547**	**0.010357**
9	**362880**	**359536.872842**	**3343.127158**	**0.009213**
10	**3628800**	**3598695.618741**	**30104.381259**	**0.008296**

Your numbers may be slightly different depending on the computer system and the precision used. Judging from the results, does the accuracy increase or decrease with increasing *n*.