Tarea 1

Elaborado por Daniela Pinto Veizaga

Fecha de elaboración: 22 de enero de 2020

Clave Única: 191471

Instrucciones: Terminar el ejercicio de las notas I/1.2 completo (son 4 ejercicios en total, dependiendo si les tocó que fueran pares o nones).

Ejercicio 1: Determina el rango de enteros de base 10 que puede representarse en una computadora de 16 bits utilizando el primer bit para el signo

In [37]:
```
//%cflags:-lm
#include<stdio.h>
#include<math.h>
int main(){
    printf("El rango de enteros de base 10 que puede representarse en una computadora de  16 bits utilizando el primer bit para el signo es: %f",(pow(2,15)+pow(2,14)+pow(2,13)+pow(2,12)+pow(2,11)+pow(2,10)+pow(2,9)+pow(2,8)+pow(2,7)+ pow(2,6)+pow(2,5)+pow(2,4)+pow(2,3)+pow(2,2)+pow(2,1)+pow(2,0))*2);
    return 0;
}
```

```
El rango de enteros de base 10 que puede representarse en una computadora de  16 bits utilizando el primer bit para el signo es: 131070.000000
```

Ejercicio 2 y 3: Da más ejemplos de ejecuciones que resulten en **overflow** y **underflow**.

El **número de máquina normalizado** más pequeño positivo es:

$$2^{-1022}(1+0) \approx 2.2251 \times 10^{-308}$$

Los numéros que tiene una magnitud menor a

$$2^{-1023}(1+2^{-52}) \approx 10^{-308}$$

resultan en **underflow**.

Los números mayoores que

$$2^{1024}(2+2^{-52}) \approx 10^{308}$$

resultan en **overflow**.

In [19]:
```
#include<stdio.h>
#include<float.h>
int main(){
    printf("Numéro más grande positivo: %e\n", DBL_MAX);
    return 0;
}
```

```
Numéro más grande positivo: 1.797693e+308
```

Números que rebasen este límite superior resultan en un overflow:

In [27]:
```
//%cflags:-lm
#include<stdio.h>
#include<float.h>
#include<math.h>
int main(){
    printf("overflow: %e\n", DBL_MAX + 0.00000000000000001*pow(10,308))
;
    return 0;
}
```

```
overflow: inf
```

In [35]:
```
//%cflags:-lm
#include<stdio.h>
#include<float.h>
#include<math.h>
int main(){
    printf("overflow: %e\n", DBL_MAX + 0.00001*pow(10,600));
    return 0;
}
```

```
overflow: inf
```

El número de máquina normalizado más pequeño positivo es:

$$2^{-1022}(1+0) \approx 2.2251 \times 10^{-308}.$$

In [100]:
```c
#include<stdio.h>
#include<float.h>
int main(){
    printf("Numéro normalizado más chico positivo: %e\n", DBL_MIN);
    return 0;
}
```

```
Numéro normalizado más chico positivo: 2.225074e-308
```

El número de máquina no normalizado o subnormalizado más chico positivo es del orden de $2^{-1022}(2^{-52}) = 2^{-1074} \approx 10^{-324}$.

In [102]:
```c
//%cflags:-lm
#include<stdio.h>
#include<float.h>
#include<math.h>
int main(){
    printf("Numéro más chico positivo: %e\n", pow(2,-52)*DBL_MIN);
    return 0;
}
```

```
Numéro más chico positivo: 4.940656e-324
```

Números con magnitud más chica que el valor anterior resultan en un underflow: Ejercicio: da más ejemplos de ejecuciones como la anterior que resulten en underflow.

In [103]:
```c
//%cflags:-lm
#include<stdio.h>
#include<float.h>
#include<math.h>
int main(){
    printf("Underflow: %e\n",(1-.5)*pow(2,-52)*DBL_MIN);
    return 0;
}
```

```
Underflow: 0.000000e+00
```

Ejercicio 4: Considérese: $x = \frac{5}{7} \approx \overline{0.714285}$, $y = \frac{1}{3} = \overline{.3}$, $u = 0.714251$, $v = 98765.9$, $w = 0.111111 \times 10^{-4}$.

Un SPFN con:

$$\beta = 10, k = 5$$

Entonces llenar la siguiente tabla de acuerdo a las instrucciones:

1. En la columna "Aritmética de máquina con k=8" se realiza aritmética a 8 dígitos con las operaciones definidas previamente $\oplus, \ominus, \otimes, \oslash$ con la regla de corte $fl_c(\cdot)$.
2. En la columna "Aritmética de máquina con k=5" se realiza aritmética a 5 dígitos con las operaciones definidas previamente $\oplus, \ominus, \otimes, \oslash$ con la regla de corte $fl_c(\cdot)$.
3. Para el cálculo de errores absoluto y relativo tomar como valor real u objetivo el valor a 8 dígitos.

In [98]:
```
//%cflags:-lm
#include<stdio.h>
#include<math.h>

int main(){

    /*Definición de variables*/
    double x = 5.0/7.0;
    double y = 1.0/3.0;
    double u = 0.714251;
    double v = 98765.9;
    double w = 0.111111e-4;
    double x_t, y_t, u_t, v_t, w_t;
    double a8, a5, err_a, err_r;


    x_t = truncf(x*10000)/10000;
    y_t = truncf(y*10000)/10000;
    u_t = truncf(u*10000)/10000;
    v_t = truncf(v*10000)/10000;
    w_t = truncf(w*10000)/10000;

    /*header de la tabla*/
    printf("Operación \t A.M. (k =8) \t A.M (k =5) \tError Absoluto\t
Error Relativo\n");

    /* operación: 'x + y' */
    a8 = x + y;
    a8 = truncf(a8*10000000)/10000000;
    a5 = x_t + y_t;
    a5 = truncf(a5*10000)/10000;
    err_a = a8 - a5;

    printf("x + y\t\t");
    printf("%1.7e\t", a8);
```

```
printf("%1.4e\t", a5);
printf("%e\t", err_a);
printf("%e\n",  err_a/a8);

/* operación: 'x - y' */
a8 = x - y;
a8 = truncf(a8*10000000)/10000000;
a5 = x_t - y_t;
a5 = truncf(a5*10000)/10000;
err_a = a8 - a5;

printf("x - y\t\t");
printf("%1.7e\t", a8);
printf("%1.4e\t", a5);
printf("%e\t", err_a);
printf("%e\n",  err_a/a8);


/* operación: 'x * y' */
a8 = x * y;
a8 = truncf(a8*10000000)/10000000;
a5 = x_t * y_t;
a5 = truncf(a5*10000)/10000;
err_a = a8 -a5;

printf("x * y\t\t");
printf("%1.7e\t", a8);
printf("%1.4e\t", a5);
printf("%e\t", err_a);
printf("%e\n",  err_a/a8);

/* operación: 'x / y' */
a8 = x / y;
a8 = truncf(a8*10000000)/10000000;
a5 = x_t / y_t;
a5 = truncf(a5*10000)/10000;
err_a = a8 -a5;

printf("x / y\t\t");
printf("%1.7e\t", a8);
printf("%1.4e\t", a5);
printf("%e\t", err_a);
printf("%e\n",  err_a/a8);

/* operación: 'x - u' */
a8 = x - u;
a8 = truncf(a8*10000000)/10000000;
a5 = x_t - u_t;
a5 = truncf(a5*10000)/10000;
err_a = a8 - a5;

printf("x - u\t\t");
printf("%1.7e\t", a8);
```

```
    printf("%1.4e\t", a5);
    printf("%e\t", err_a);
    printf("%e\n",  err_a/a8);

    /*operación: '(x - u)/w' */
    a8 = x - u;
    a8 = truncf(a8*10000000)/10000000;
    a8 = a8 / w;
    a8 = truncf(a8*10000000)/10000000;
    a5 = x_t - u_t;
    a5 = truncf(a5*10000)/10000;
    a5 = a5 / w_t;
    a5 = truncf(a5*10000)/10000;
    err_a = a8 - a5;

    printf("(x - u)/w\t");
    printf("%1.7e\t", a8);
    printf("%1.4e\t\t", a5);
    printf("%e\t", err_a);
    printf("\t%e\n",  err_a/a8);

    /*operación: '(x - u)v' */
    a8 = x - u;
    a8 = truncf(a8*10000000)/10000000;
    a8 = a8 * v;
    a8 = truncf(a8*10000000)/10000000;
    a5 = x_t - u_t;
    a5 = truncf(a5*10000)/10000;
    a5 = a5 * v_t;
    a5 = truncf(a5*10000)/10000;
    err_a = a8 - a5;

    printf("(x - u)v\t");
    printf("%1.7e\t", a8);
    printf("%1.4e\t", a5);
    printf("%e\t", err_a);
    printf("%e\n",  err_a/a8);


    /* operación: 'u + v' */
    a8 = u + v;
    a8 = truncf(a8*10000000)/10000000;
    a5 = u_t + v_t;
    a5 = truncf(a5*10000)/10000;
    err_a = a8 -a5;

    printf("u + v\t\t");
    printf("%1.7e\t", a8);
    printf("%1.4e\t", a5);
    printf("%e\t", err_a);
    printf("%e\n",  err_a/a8);
}
```

Operación	A.M. (k =8)	A.M (k =5)	Error Absoluto	Error Relativo
x + y	1.0476190e+00	1.0475e+00	1.189709e-04	1.135631e-04
x - y	3.8095230e-01	3.8090e-01	5.230308e-05	1.372956e-04
x * y	2.3809519e-01	2.3800e-01	9.518862e-05	3.997923e-04
x / y	2.1428573e+00	2.1428e+00	5.722046e-05	2.670288e-05
x - u	3.4699999e-05	0.0000e+00	3.470000e-05	1.000000e+00
(x - u)/w	3.1230030e+00	-nan	-nan	-nan
(x - u)v	3.4271765e+00	0.0000e+00	3.427176e+00	1.000000e+00
u + v	9.8766609e+04	9.8767e+04	0.000000e+00	0.000000e+00

Ejercicio 5: Utiliza la notación posicional para representar al número: 86409 y $(1001.1)_2$ en base 10

$$86409 = 80000 + 6000 + 400 + 9$$

$$(1001.1)_2 = (1 * 2^3) + (0 * 2^2) + (0 * 2^1) + (1 * 2^0) + (1 * 2^{-1}) = (9.5)_{10}$$

¿Cuáles de los siguientes números son números de máquina en un SPFN con $\beta = 2$?

Los números reales que tienen una representación exacta en el $\mathcal{F}l$ se les conoce como números de máquina.

a. $(2.125)_{10}$.

Parte entera:

$$2/2 = 1; Residuo: 0$$

Parte decimal:

$$0.125 * 2 = 0.250$$
$$0.25 * 2 = 0.5$$
$$0.5 * 2 = 1$$

Entonces:

$$2.125 = (10.001)_2$$

b. $(3.1)_{10}$.

Parte entera:

$$3/2 = 1; Residuo = 1$$

Parte decimal:

$$0.1 * 2 = 0.2$$
$$0.2 * 2 = 0.4$$
$$0.4 * 2 = 0.8$$
$$0.8 * 2 = 1.6$$
$$0.6 * 2 = 1.2$$
$$0.2 * 2 = 0.4$$
$$0.4 * 2 = 0.8$$
$$0.8 * 2 = 1.6$$

. . .

Entonces:

$$(3.1)_{10} \approx 11.00011001$$

Por lo tanto, el número del inciso a es un **número de máquina**.

Ejercicio 6: Considérese un SPFN con $\beta = 2, k = 3$. En este sistema se tienen 7 bits para almacenar números. El primer bit se utiliza para el signo del número, el segundo bit se utiliza para el signo del exponente, los dos siguientes bits para construir al exponente y el último bit para construir a la mantisa. Entonces el número positivo normalizado más pequeño que es posible representar en este SPFN es:

$$(0111100)_2 = (.5 \times 2^{-3})_{10} = .0625.$$

a. Escribe los siguientes números más grandes a este número que forman al SPFN hasta el valor más grande positivo que es posible representar en este SPFN.

b. Calcula las distancias entre los números con mismo valor de exponente.

c. Verifica que el error relativo para $x = 0.156249$ utilizando la regla de corte es menor o igual a $\epsilon_{maq} = .25$ y el error relativo para $x = 0.14$ utilizando la regla de redondeo es menor o igual a $\epsilon_{maq} = 0.125$.

In [ ]:

Nota: todos los ejercicios fueron ejecutados empleando el presente jupyterlab con un kernell de C