Carátula para entrega de prácticas	
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación
salas A y B

Profesor: Alejandro Esteban Pimentel Alarcon

Asignatura: Fundamentos de programacíon.

Grupo: 3

No de Práctica(s): 10

Integrante(s): Crail Ávila Regina
Ortiz Garcia Cesar Alan

No. de Equipo de cómputo empleado: 20,21

No. de Lista o Brigada: 8973
9070

Semestre: 20-21

Fecha de entrega: 27/oct/19

Observaciones:

CALIFICACIÓN: _____

Primeros pasos con GDB

jjgrodriguez@gmail.com

Este documento pretende ser una breve y sencilla guía práctica al depurador GDB (http://gnu.org). A lo largo de esta guía, usaremos como ejemplo un sencillo programa cuyo código se muestra a continuación:

```
#include <stdio.h>
int main() {
 int v = 0;
   int i;
 for (i = 0; i < 5; i++) {
   v += i;
 }
 printf("Resultado: %i\n", v);
 return 0;
}
```

Lo primero que hemos de hacer es compilar dicho programa con la opción "-g" para que se incluya la información necesaria para la depuración. Si el programa está guardado en un archivo

```
$ gcc -g -o test test.c
```

Una vez compilado, podemos comenzar la depuración invocando a gbd con el nombre del archivo compilado.

```
$ gdb test
  GNU gdb 5.3-debian
  Copyright 2002 Free Software Foundation, Inc.
  GDB is free software, covered by the GNU General Public License, and you are
  welcome to change it and/or distribute copies of it under certain conditions.
  Type "show copying" to see the conditions.
  There is absolutely no warranty for GDB. Type "show warranty" for details.
  This GDB was configured as "i386-linux"...
  (gdb)
```

La línea "(gdb)" nos indica que el depurador está listo para empezar a procesar nuestras órdenes. El primer comando que veremos es (l)ist. Este comando muestra el código fuente del programa que estoy depurando.

```
(gdb) l
  1 #include <stdio.h>
  2
  3 int main() {
  4
  5 int v = 0;
  6 int i;
  7
  8 for (i = 0; i < 5; i++) {
  9 v += i;
  10 }
(gdb)
```

Para ejecutar el programa, utilizamos el comando (r)un. Como aún no hemos puesto ningún punto de ruptura, el programa se ejecutará entero.

```
(gdb) r
  Starting program: /ramdisk/home/knoppix/test
  Resultado: 10
  Program exited normally.
(gdb)
```

Vamos a colocar ahora un punto de ruptura. Como su nombre indica, un punto de ruptura es un punto donde la ejecución de nuestro programa se detiene. Para colocar un punto de ruptura utilizamos el comando (b)reakpoint seguido de la línea donde queremos ponerlo.

```
(gdb) b 9
  Breakpoint 1 at 0x804834e: file test.c, line 9.
(gdb)
```
Ya tenemos un punto de ruptura en la línea 9. Si ahora ejecutamos el programa se detendrá cuando llegue a esa línea.
```
(gdb) r
  Starting program: /ramdisk/home/knoppix/test
  Breakpoint 1, main () at test.c:9
  9 v += i;
(gdb)
```
Para continuar la ejecución del programa utilizamos el comando (c)ontinue. En nuestro ejemplo, el programa se volverá a detener en el mismo punto de ruptura.
```
(gdb) c
  Continuing.
  Breakpoint 1, main () at test.c:9
  9 v += i;
(gdb) c
  Continuing.
  Breakpoint 1, main () at test.c:9
  9 v += i;
(gdb) c
  Continuing.
  Breakpoint 1, main () at test.c:9
  9 v += i;
(gdb) c
  Continuing.
  Breakpoint 1, main () at test.c:9
  9 v += i;
(gdb) c
  Continuing.
  Resultado: 10
  Program exited normally.
(gdb)
```
Puesto que tenemos un punto de ruptura en el bucle y este da 5 vueltas, hemos tenido que continuar 5 veces para ejecutar el programa completo. También es posible ejecutar el programa línea a línea con el comando (n)ext.
```
(gdb) r
  Starting program: /ramdisk/home/knoppix/test
  Breakpoint 2, main () at test.c:9
  9 v += i;
  (gdb) n
  8 for (i = 0; i < 5; i++) {
  (gdb) n
  Breakpoint 2, main () at test.c:9
  9 v += i;
(gdb)
```
En cualquier momento podemos ver el valor de una variable con el comando (p)rint seguido del nombre de la variable.
```
(gdb) p v
  $1 = 1
  (gdb) c
  Continuing.
  Breakpoint 2, main () at test.c:9
  9 v += i;
(gdb) c
  Continuing.
```

```
  Breakpoint 2, main () at test.c:9
  9 v += i;
(gdb) p v
  $2 = 6
(gdb)
```

Para eliminar un punto de ruptura utilizamos el comando delete y el número del punto a eliminar. En nuestro ejemplo

```
(gdb) delete 1
(gdb)
```

Por último, para salir del depurador utilizamos el comando (q)uit.

Hay mucha más información disponible tanto en la página del manual como en los archivos info:

```
$ man gdb
$ info gdb
```

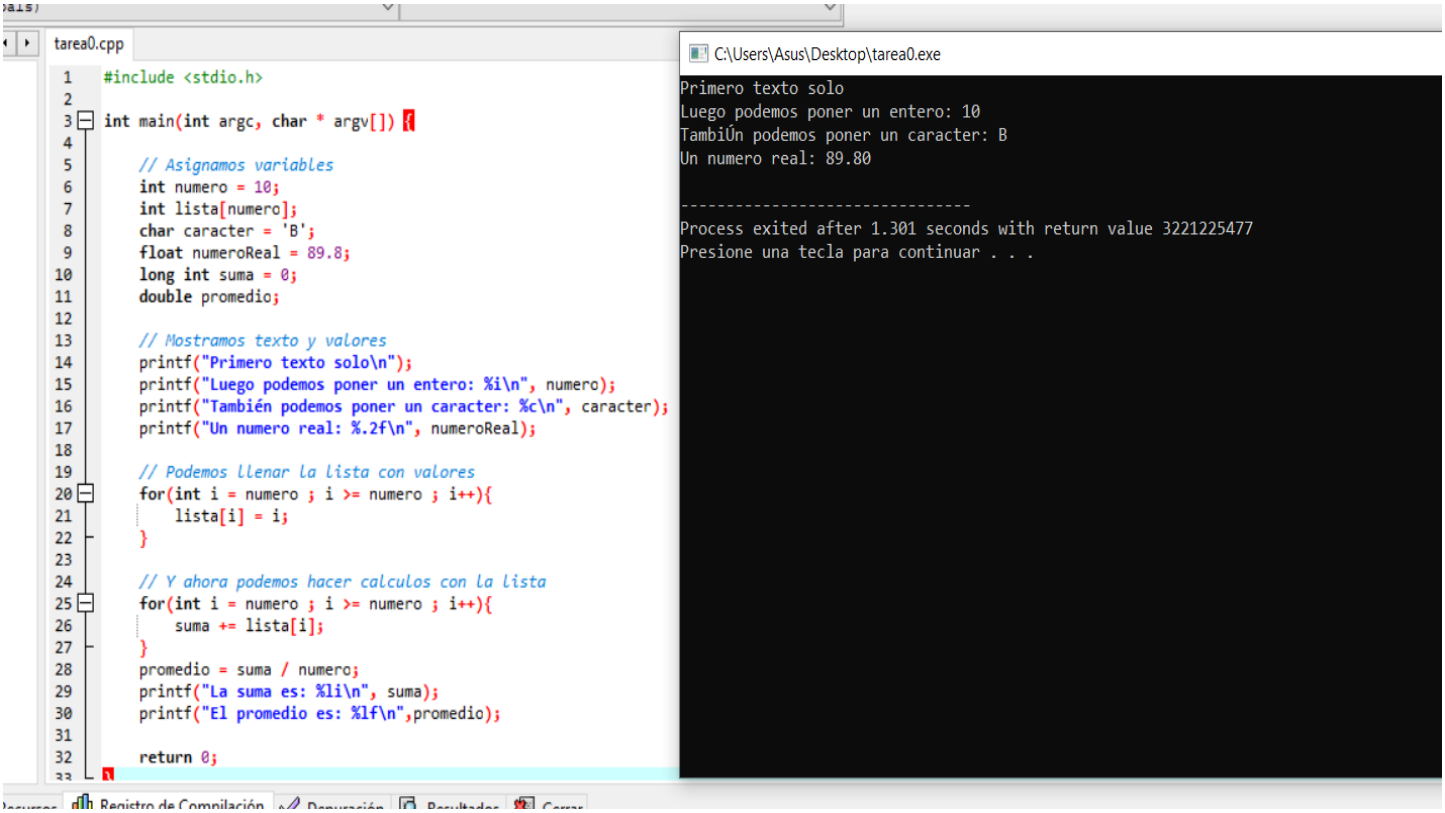
También puede utilizarse interfaces gráficos para trabajar con GB de manera más cómoda. Uno de esos interfaces es DD y puede descargarse en http://sourceforge.net/projects/ddd/

1._

- Utilizar GDB para encontrar la utilidad del programa y describir su funcionalidad.

Se ejecutó y probó el programa adecuadamente, usando comandos, samba, se pudo llegar a que el primer programa no tenía ningún error y se podía correr bien.

Lo que hace es que recibe un número, después otro, se guarda el resultado de la suma más el contador así que aumenta de dos en dos y inicia desde1.



tarea0.cpp

```
#include <stdio.h>

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

    // Asignamos variables
    int numero = 10;
    int lista[numero];
    char caracter = 'B';
    float numeroReal = 89.8;
    long int suma = 0;
    double promedio;

    // Mostramos texto y valores
    printf("Primero texto solo\n");
    printf("Luego podemos poner un entero: %i\n", numero);
    printf("También podemos poner un caracter: %c\n", caracter);
    printf("Un numero real: %.2f\n", numeroReal);

    // Podemos llenar la lista con valores
    for(int i = numero ; i >= numero ; i++){
        lista[i] = i;
    }

    // Y ahora podemos hacer calculos con la lista
    for(int i = numero ; i >= numero ; i++){
        suma += lista[i];
    }
    promedio = suma / numero;
    printf("La suma es: %li\n", suma);
    printf("El promedio es: %lf\n",promedio);

    return 0;
}
```

C:\Users\Asus\Desktop\tarea0.exe

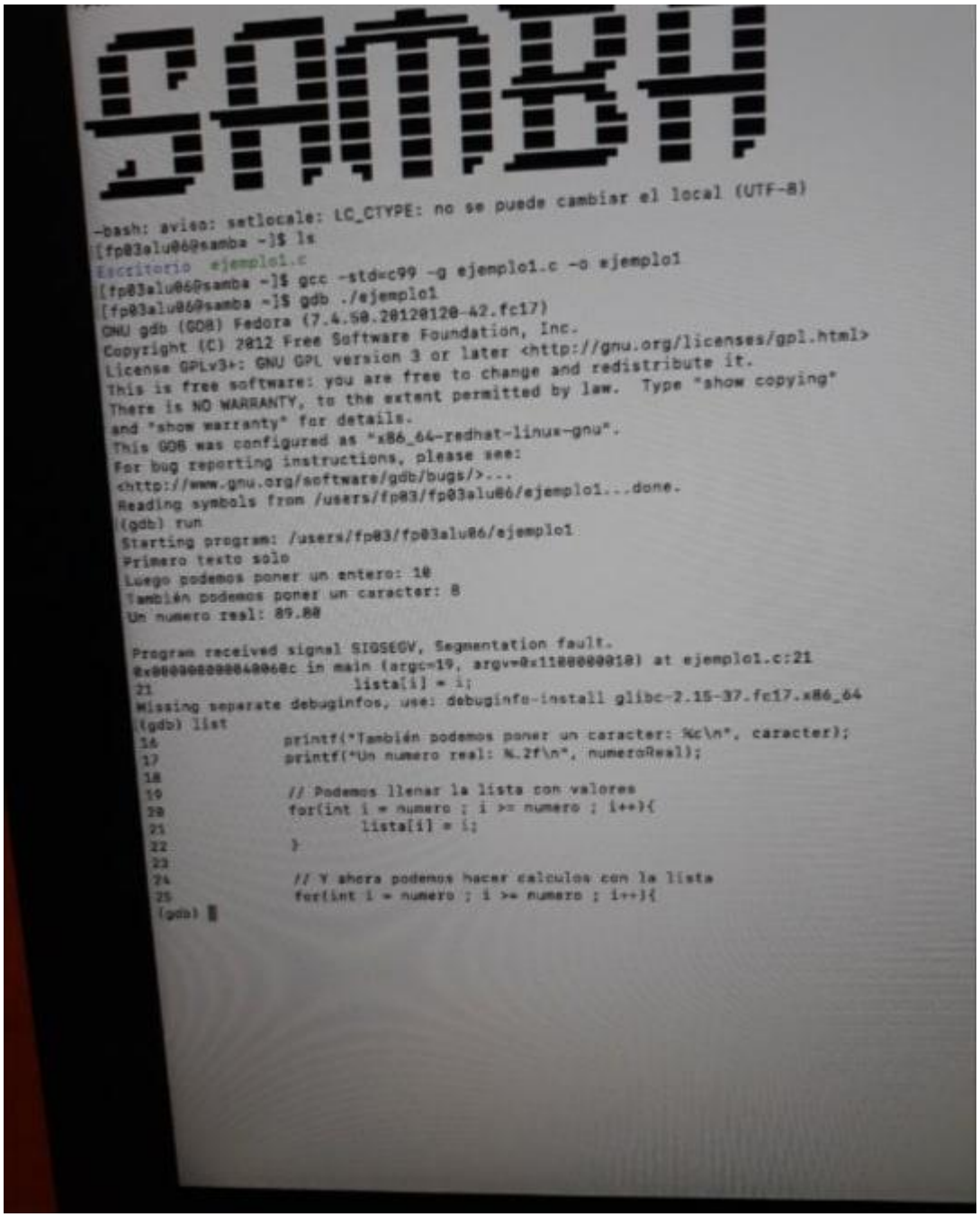
```
Primero texto solo
Luego podemos poner un entero: 10
TambiÚn podemos poner un caracter: B
Un numero real: 89.80

--------------------------------
Process exited after 1.301 seconds with return value 3221225477
Presione una tecla para continuar . . .
```

rsos | Registro de Compilación | Depuración | Resultados | Cerrar

```
Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\Asus\Desktop\tarea0.exe
- Output Size: 128.6005859375 KiB
- Compilation Time: 0.27s
```

Sel: 0 Lines: 33 Length: 807 Insertar Done parsing in 0.047 seconds

uí para buscar



```
21                      lista[i] = i;
Missing separate debuginfos, use: debuginfo-install glibc-2.15-37.fc17.x86_64
(gdb) list
16              printf("También podemos poner un caracter: %c\n", caracter);
17              printf("Un numero real: %.2f\n", numeroReal);
18
19              // Podemos llenar la lista con valores
20              for(int i = numero ; i >= numero ; i++){
21                      lista[i] = i;
22              }
23
24              // Y ahora podemos hacer calculos con la lista
25              for(int i = numero ; i >= numero ; i++){
(gdb) quit
A debugging session is active.

        Inferior 1 [process 22276] will be killed.

Quit anyway? (y or n) y
[fp03alu37@samba ~]$ gdb ejemplo1
GNU gdb (GDB) Fedora (7.4.50.20120120-42.fc17)
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /users/fp03/fp03alu37/ejemplo1...done.
(gdb) break 20
Breakpoint 1 at 0x4005f7: file ejemplo1.c, line 20.
(gdb) run
Starting program: /users/fp03/fp03alu37/ejemplo1
Primero texto solo
Luego podemos poner un entero: 10
También podemos poner un caracter: B
Un numero real: 89.80

Breakpoint 1, main (argc=1, argv=0x7fffffffe398) at ejemplo1.c:20
20              for(int i = numero ; i >= numero ; i++){
Missing separate debuginfos, use: debuginfo-install glibc-2.15-37.fc17.x86_64
(gdb) n
21                      lista[i] = i;
(gdb) print lista
$1 = {-163754450, 0, 4195102, 0, -1, 0, -7536, 32767, -7520, 32767}
(gdb) display i
1: i = 10
(gdb) dsplay lista
Undefined command: "dsplay".  Try "help".
(gdb) n
20              for(int i = numero ; i >= numero ; i++){
1: i = 10
(gdb) n
21                      lista[i] = i;
1: i = 11
(gdb) n
20              for(int i = numero ; i >= numero ; i++){
1: i = 11
(gdb) n
21                      lista[i] = i;
1: i = 12
(gdb) n
20              for(int i = numero ; i >= numero ; i++){
1: i = 12
(gdb) n
21                      lista[i] = i;
1: i = 13
(gdb) n
20              for(int i = numero ; i >= numero ; i++){
1: i = 13
(gdb) n
21                      lista[i] = i;
1: i = 14
(gdb) n
20              for(int i = numero ; i >= numero ; i++){
```

2._ CORREGIR

```
#include <stdio.h>
#include <math.h>

void main()
{
        int K, AP, N;
        double X, AS;
        printf("Ingrese cuántos términos calcular de la serie: X^K/K!");
        printf("\nN=");
        scanf("%i",N);
        printf("X=");
        scanf("%lf",X);
        K=0;
        AP=1;
        AS=0;
        while(K<=N)
        {
                AS=AS+pow(X,K)/AP;
                K=K+1;
                AP=AP*K;
        }
        printf("Resultado=%le",AS);
}
```

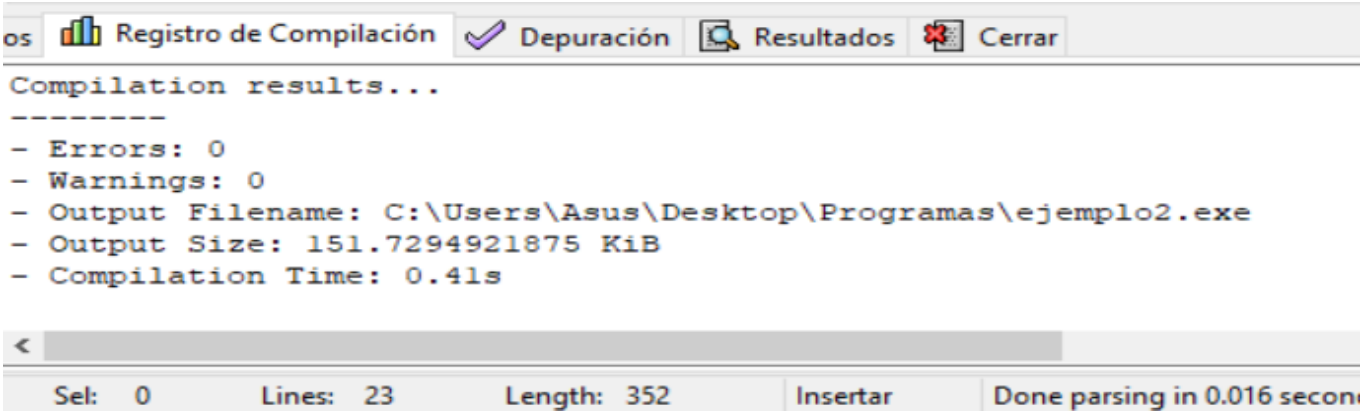
Ya que para nosotros se nos hace más práctico y accesible usar C++ tanto para la corrección de programas como la ejecución lo decidimos usar puesto que es MÁS CLARO y sin tantas tantas cosas que para nosotros es innecesario, analizando y haciendo correcciones que nos marcaba c++ las cambiamos, y este fue el resultado al que llegamos.

La ejecución hace un EXPONENCIAL, dos números ingresados, X, Y.

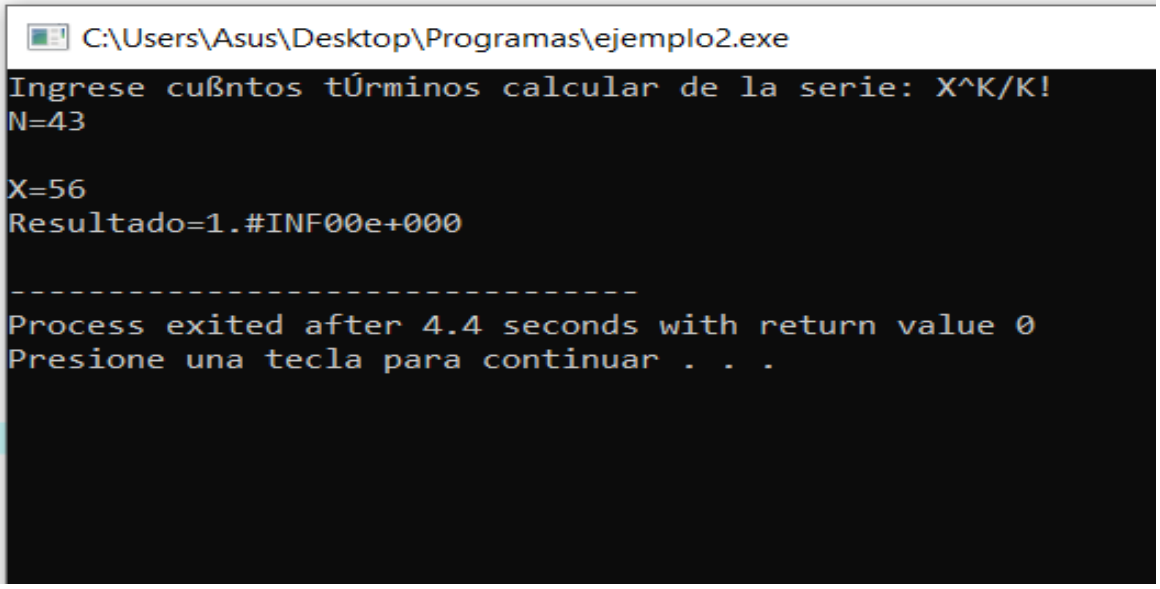
CORREGIDO.

ejemplo2.cpp

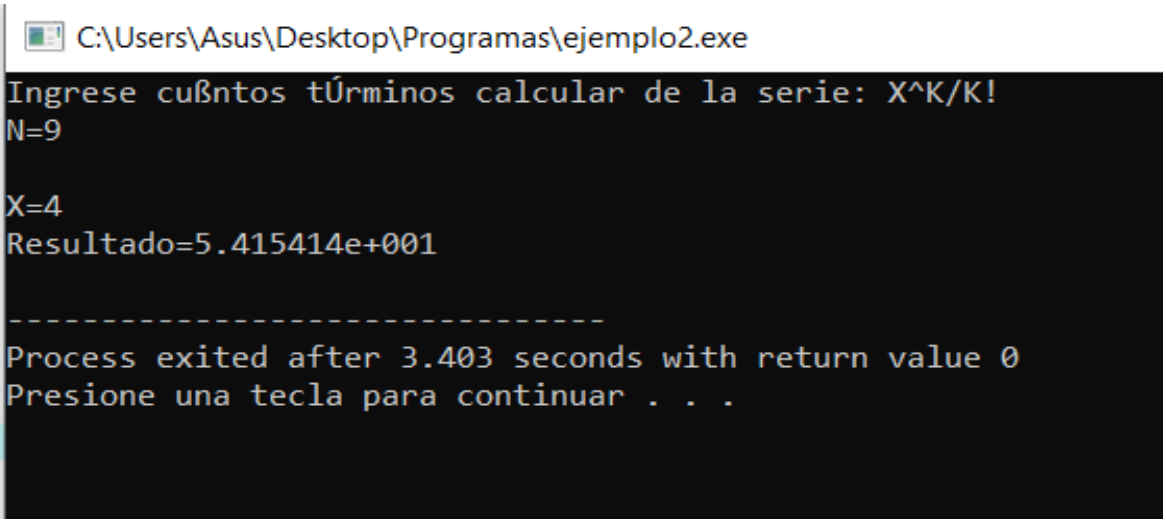
```
1   #include <stdio.h>
2   #include <math.h>
3
4   int main()
5   {
6       int K, AP, N;
7       double X, AS;
8       printf("Ingrese cuántos términos calcular de la serie: X^K/K!");
9       printf("\nN=");
10      scanf("%i",&N);
11      printf("\nX=");
12      scanf("%lf",&X);
13      K=0;
14      AP=1;
15      AS=0;
16      while(K<=N)
17      {
18          AS=AS+pow(X,K)/AP;
19          K=K+1;
20          AP=AP*K;
21      }
22      printf("Resultado=%le\n",AS);
23  }
```




```
s términos calcular de la serie: X^K/K!");
```




```
n",AS);
```



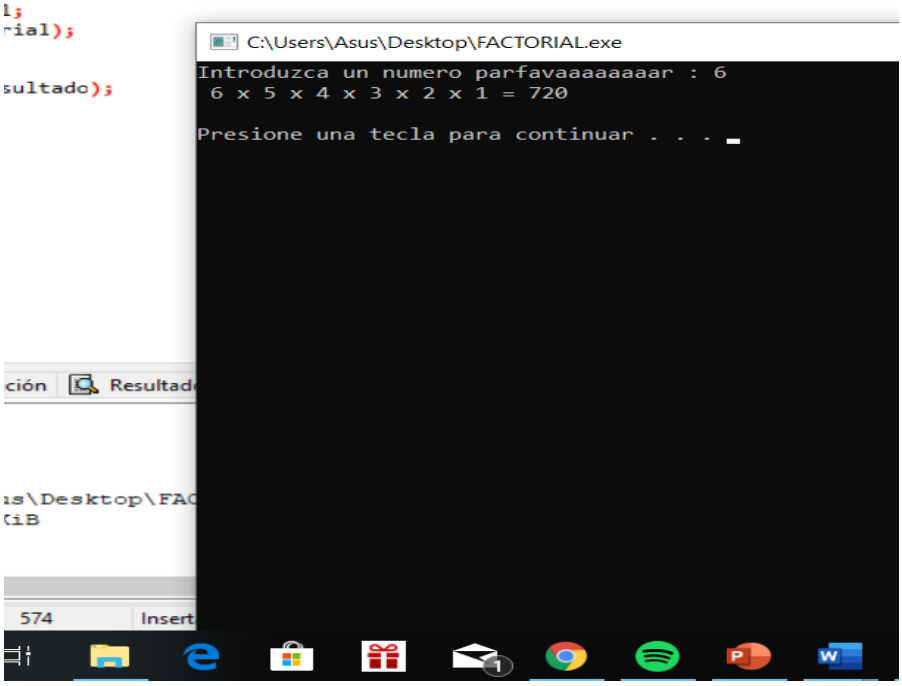

```
';
```

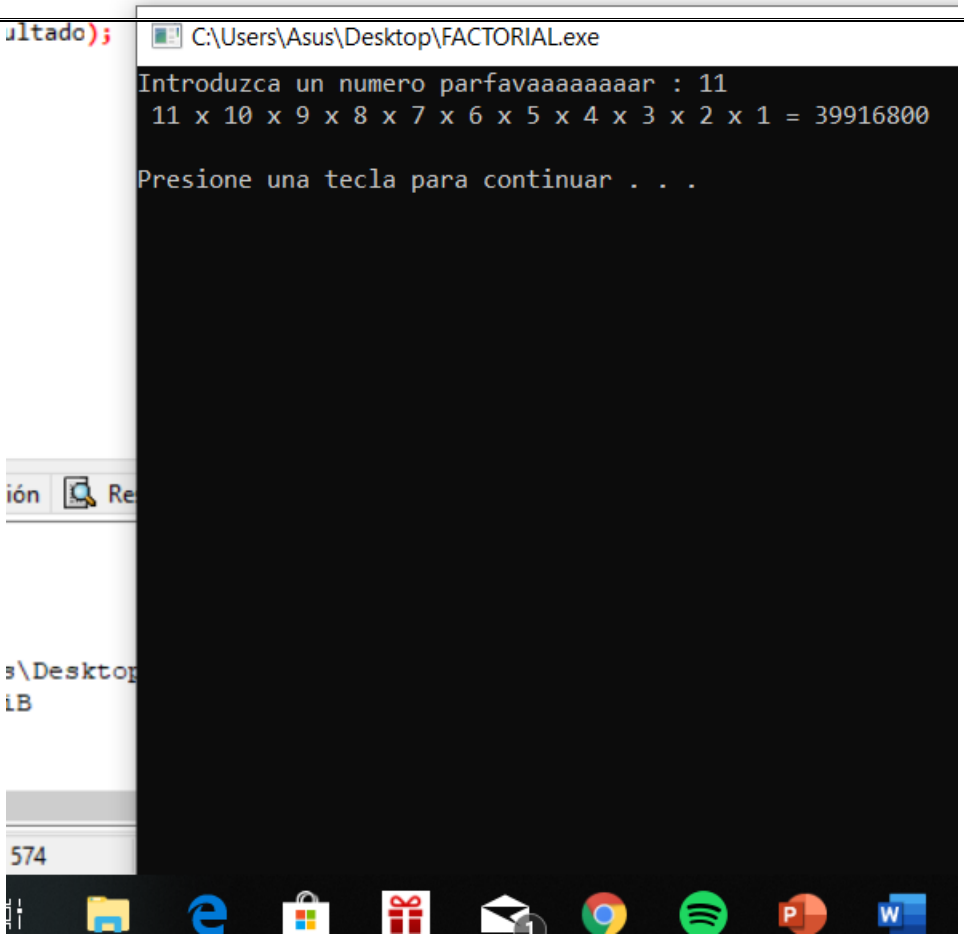
```
:\n",AS);
```

3._ CORREGIR.

```
#include <stdio.h>

int main()
{
        int numero;

        printf("Ingrese un número:\n");
        scanf("%i",&numero);

        long int resultado = 1;
        while(numero>=0){
                numero--;
                resultado *= numero;
        }

    printf("El factorial de %i es %li.\n", numero, resultado);

    return 0;
}
```

Haciendo uso de c++ de igual forma a las pasadas del misto puesto en la documentación de la misma introducida con anterioridad actual, se corrigió el programa, ya que tuvimos que hacer y cambiar varias que nos marcaba c++, el programa es para calcular la factorial de un número





CONCLUSIONES:

Es mucho más fácil y práctico usar c++ ya que no da tantos rodeos y te lo dice claro y conciso, cada programa es diferente pero tiene fallas, las cuales igual con habilidad se pudieron haber corregido, aún así amamos c++ a comparación de otro editores ya que los demás están pésimos.