Universidad de Buenos Aires
Facultad De Ingeniería
Año 2021 - 1.er Cuatrimestre

# Algorítmos y Programación II (95.12)

## Trabajo Práctico N°0


| | | |
|---|---|---|
| Berard, Lucía Magdalena | 101213 | lberard@fi.uba.ar |
| Guglieri, Mariano Federico | 99573 | mguglieri@fi.uba.ar |
| Rubin, Ivan Eric | 100577 | irubin@fi.uba.ar |
| Sandoval, Diego Ariel | 101639 | dsandoval@fi.uba.ar |

## Índice

## 1. Introducción

El objetivo fundamental del trabajo es implementar números enteros de precisión fija (pero arbitrariamente grande). Se entiende como precisión a la cantidad de dígitos decimales que conforman el número.

En una primera etapa, se desea representar este concepto utilizando arreglos. Para ello, se desarrolló una clase C++ `bignum` siguiendo el esquema general que se muestra a continuación:

```
class bignum
{
     private:
          unsigned short *digits;
          // ...
     public:
          // ...
          friend bignum operator+(const bignum&, const bignum&);
          friend bignum operator-(const bignum&, const bignum&);
          friend bignum operator*(const bignum&, const bignum&);
          friend std::ostream& operator<<(std::ostream&, const bignum&);
          friend std::istream& operator>>(std::istream&, bignum&);
};
```

Como se observa en este fragmento, es preciso además sobrecargar los operadores aritméticos de suma, resta y multiplicación y los de entrada y salida con formato. Las porciones marcadas con puntos suspensivos corresponden a los restantes métodos y/o variables necesarias para el correcto funcionamiento del programa.

### 1.1. Línea de comando

Las opciones `-i` y `-o` permiten seleccionar los streams de entrada y salida respectivamente. Por defecto, éstos son `cin` y `cout`. Lo mismo ocurre al recibir `\-"` como argumento.

Por otro lado, la opción `-p` indica el valor de la precisión con la que se lleva a cabo el procesamiento. Puede asumirse que los números ingresados en el stream de entrada se ajustan a dicha precisión, aunque podría ocurrir que el resultado de alguna de las expresiones de entrada requiera mayor cantidad de dígitos.

Al finalizar, todos los programas retornan un valor nulo en caso de no detectar ningún problema; en caso contrario, devuelven un valor no nulo
En la sección 4, ejecución del programa, se detallará más el uso de los comandos de línea.

### 1.2. Formato de los archivos de entrada y salida

El formato a adoptar para el stream de entrada consiste en una secuencia de cero o más expresiones aritméticas binarias, cada una ocupando una línea distinta. A su vez, la separación entre cada operando y el operador puede darse con cero o más espacios, entendiendo por tales a los caracteres `SP, \f}, \r, \t, \v.`

Por otro lado, el stream de salida deberá listar en líneas distintas los números resultantes de evaluar cada expresión, manteniendo el mismo orden de las operaciones de la entrada.

## 2. Diseño e implementación del programa

Para la creación de la clase `bignum`, al tratarse de un número entero de precisión fija pero arbitrariamente grande, se la pensó como un vector de `unsigned short` que contenga los dígitos, un `bool` que contenga al signo del número y un `unsigned short` para la longitud del vector.

En cuanto a las operaciones propias de la clase se desarrollaron las pedidas por el enunciado y se agregaron otras. Las otras funciones desarrolladas se crearon con el propósito de facilitar la creación de las operaciones básicas de suma resta y multiplicación. Un ejemplo de estas operaciones son la sobrecarga de los operadores mayor, menor y el operador igualdad. Además, se crearon funciones para darle formato a los números, como la función para setear el largo, la precisión y el signo. Se crearon también 3 diferentes constructores para facilitar la creación de bignums auxiliares en las operaciones. A continuación se muestran las funciones que se pedían en el desarrollo del trabajo:

```
1   class bignum
2   {
3     private:
4       unsigned short *digits;
5       unsigned short len;
6       bool signo; //True: positivo - False: negativo
7
8     public:
9       //Constructores
10      bignum(void);
11      bignum(const unsigned short);
12      bignum(const string&);
13      //Destructor
14      ~bignum();
15
16      //Funciones útiles
17      void set_signo(bool);
18      bool get_signo();
19      unsigned char get_len();
20      void set_precision(const unsigned short);
21      friend bignum operator-(const bignum&);
22
23      //Sobrecarga de los operadores de suma, resta y multiplicación
24      friend bignum operator+(const bignum&, const bignum&);
25      friend bignum operator-(const bignum&, const bignum&);
26      friend bignum operator*(const bignum&, const bignum&);
27
28      //Sobrecarga del operador de asignación
29      const bignum& operator=(const bignum&);
30      const bignum& operator=(const string&);
31
32      //Sobrecarga de los operadores de comparación
33      friend bool operator==(const bignum&, const bignum&);
34      friend bool operator<(const bignum&, const bignum&);
35      friend bool operator>(const bignum&, const bignum&);
36
37      //Sobrecarga de los operadores de flujo de stream
38      friend std::ostream& operator<<(std::ostream&, const bignum&);
39      friend std::istream& operator>>(std::istream&, bignum&);
40  };
```

Listing 1: Declaración de la clase bignum

Como se puede observar en el código, se desarrollaron 3 constructores para poder crear un bignum vacío, uno a partir de una determinada longitud y otro a partir de un string:

```
1  bignum::bignum(void)
2  {
3    signo = true;
4    len = 10;
5    digits = new unsigned short[len];
6
7    for(size_t i=0;i<len;i++)
8    {digits[i]=0;}
9  }
```

Listing 2: Constructor de un bignum vacío de longitud por default de 10

```
1  bignum::bignum(const unsigned short a)
2  {
3    signo = true;
4    len = a;
5    digits = new unsigned short[len];
6    for(size_t i=0;i<len;i++)
7    {digits[i]=0;}
8  }
```

Listing 3: Constructor de un bignum a partir de una longitud determinada

```
bignum::bignum(const string &str1)
{

  //Saco los espacios en blanco.
  string str;
  for(char c:str1) if(!isspace(c)) str += c ;

  if(!(str.find_first_not_of("0123456789") == string::npos) && (str[0]!='-' && str[0]!='+'))
  {
    cerr<<"Asignacion de numero invalida"<<endl;
    exit(1);
  }

  //Defino el signo.
  bool hay_signo;
  if(str[0]=='-')
  {
    signo=false;
    hay_signo=true;
    len=str.length()-1;
  } else if(str[0]=='+')
  {
    signo=true;
    hay_signo=true;
    len=str.length()-1;
  }else{
    hay_signo=true;
    len=str.length();
  }
  //Creo el arreglo de shorts
  digits=new unsigned short[len];


  for(size_t i=0;i<len;i++)
  {
    digits[len-1-i]=str[len+hay_signo-i]-ASCII_FIX;
  }
}
```

Listing 4: Constructor de un bignum a partir de un string

y el correspondiente destructor que se debe utilizar para evitar fallas de memoria:

```
bignum::~bignum()
{
  if(digits)
  {delete[] digits;}
}
```

Listing 5: Constructor de un bignum a partir de un string

Para la resolución de las operaciones, al tratarse de números en formato de arrays se solucionó pensándolo de la siguiente manera:

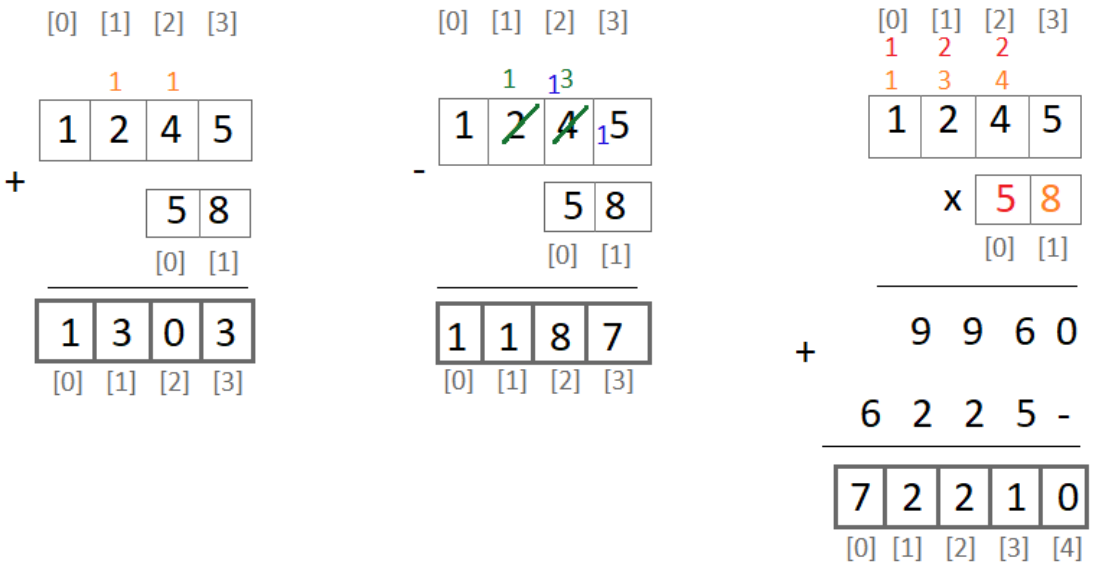


Figura 2.1: Operaciones

Se separaron las operaciones por dígito, uniéndolas luego en el resultado final, teniendo en cuenta los casos específicos, tal como se muestra en la figura.

```
bignum operator+(const bignum& a, const bignum& b)
{

  unsigned short n = a.len;
  unsigned short m = b.len;
  bignum result(max(n,m)+1);
  if(!a.signo && b.signo){
    bignum c = -a;
    result=b-c;
    return result;
  }
  if(a.signo && !b.signo){
    bignum c = -b;
    result=a-c;
    return result;
  }
  if(!a.signo && !b.signo){
    result.signo=false;
  }
  unsigned short carry = 0;
  unsigned short aux = 0;

  for(int i=0; i< max(n,m)+1;i++){

    if((short)(m-i-1)<0 && (short)(n-i-1)<0){
      aux=carry;
    } else if((short)(n-i-1)<0){
      aux=b.digits[m-i-1]+carry;
    } else if((short)(m-i-1)<0){
      aux=a.digits[n-i-1]+carry;
    } else {
      aux=a.digits[n-i-1]+b.digits[m-i-1]+carry;
    }

    if(aux>=10){
      aux-=10;
      carry=1;
    } else {
      carry=0;
    }
    result.digits[max(n,m)-i]=aux;
    result.digits[0] = aux;
  }

  return result;
}
```

Listing 6: Sobrecarga del operador suma para la clase bignum

```
bignum operator-(const bignum& a, const bignum& b)
{

  unsigned short n = a.len;
  unsigned short m = b.len;
  bignum result(max(n,m));


  if(a==b){
    return result;
  }
  if(a<b){
    result = b-a;
    result.signo=false;
    return result;
  }
  if(!a.signo && b.signo){
    bignum c=-a;
    result = c+b;
    result.signo=false;
    return result;
  }
  if(a.signo && !b.signo){
    bignum c = -b;
    result=a+c;
    return result;
  }
  if(!a.signo && !b.signo){

    bignum c= -b;
    bignum d= -a;
    result = c-d;
    return result;
  }

  short carry = 0;
  short aux = 0;

  for(size_t i=0; i< max(n,m);i++){

    if((short)(m-i-1)<0 && (short)(n-i-1)<0){
      aux=carry;
    } else if((short)(m-i-1)<0){
      aux=a.digits[n-i-1]-carry;
    } else {
      aux=(short)a.digits[n-i-1]-(short)b.digits[m-i-1]-carry;
    }


    if(aux<0){
      aux+=10;
      carry=1;
    } else {
      carry=0;
    }
    result.digits[result.len-i-1]=aux;
  }
  return result;
}
```

Listing 7: Sobrecarga del operador resta para la clase bignum

```
bignum operator*(const bignum& a, const bignum& b){

  unsigned short n = a.len;
  unsigned short m = b.len;
  unsigned short carry = 0;
  bignum aux(n+m);
  bignum aux2(n+m);
  bignum result(n+m);

  for(size_t j = 0; j < m; j++){
    for(size_t i = 0; i < n; i++){
      aux.digits[n+m-1-i-j] = a.digits[n-1-i]*b.digits[m-1-j] + carry;
      if(aux.digits[n+m-1-i-j] >= 10){
        carry = aux.digits[n+m-1-i-j]/10;
        aux.digits[n+m-1-i-j] -=  10*(aux.digits[n+m-1-i-j]/10);
      }
      else carry = 0;
    }
    aux.digits[m-1-j] = carry;

  for(size_t k = 0; k < j; k++){
    aux.digits[n+m-1-k] = 0;
  }
    aux2 = aux2 + aux;
    carry = 0;
  }

  result = aux2;
  if((!a.signo && b.signo) ||(a.signo && !b.signo) ){
    result.signo = false;
  }
  return result;
}
bool operator==(const bignum&a, const bignum&b)
{
  if(a.signo==b.signo && a.len==b.len)
  {
    for(size_t i = 0; i < a.len; i++)
    {
      if(a.digits[i]!=b.digits[i])
        return false;
    }
    return true;
  }
  return false;
}
```

Listing 8: Sobrecarga del operador multiplicación para la clase bignum

Luego, para poder observar los resultados, es necesario sobrecargar los operadores de entrada y salida:

```
ostream& operator<<(ostream& os, const bignum& num){

if(num.signo==false){
    os << '-'; }
bool aux= false;

for(int i = 0; i< num.len;i++){

//saco los ceros de la izquierda
if(num.digits[i]==0 && aux==false){
  if(i==num.len-1)
    {
        os<<'0';
        return os;}
  continue;
}
os << num.digits[i];
aux=true;
}
 return os;
}

istream& operator>>(istream& is, bignum& num){

string s;
is >> s;
while(!(s.find_first_not_of( "0123456789" ) == string::npos) && (s[0]!='-' && s[0]!='+')){
    cerr << "El valor ingresado no es correcto. Intente nuevamente." << endl;
    is >> s;
}
num = s;
return is;
}
```

Listing 9: Sobrecarga de los operadores de flujo de stream para la clase bignum

También se plantearon funciones extras útiles para el manejo de la clase, tal como:

```
1   //Setea el signo del bignum, true si es positivo y false si es negativo
2   void bignum::set_signo(bool s){
3     signo=s;
4   }
5
6   //Devuelve el signo del bignum, true si es positivo y false si es negativo
7   bool bignum::get_signo(){
8     return signo;
9   }
10
11  //Devuelve la longitud del bignum
12  unsigned char bignum::get_len(){
13    return len;
14  }
15
16  //Setea la precision del bignum.
17  //Redondea para arriba las cifras significativas y deja el resto de los valores en cero.
18  void bignum::set_precision(const unsigned short precision){
19    if(precision>0 && len>precision)
20    {
21      for(int i=0; i<len-precision; i++){
22        if(digits[precision]>=5)
23          digits[precision-1]++;
24        digits[precision+i]=0;
25      }
26    }
27  }
```

Listing 10: Funciones útiles

y la sobrecarga de los operadores booleanos de comparación:

```cpp
//Sobrecarga del operador == para la comparacion entre dos bignums
bool operator==(const bignum&a, const bignum&b)
{
  if(a.signo==b.signo && a.len==b.len)
  {
    for(size_t i = 0; i < a.len; i++)
    {
      if(a.digits[i]!=b.digits[i])
        return false;
    }
    return true;
  }
  return false;
}

//Sobrecarga del operador < para la comparacion entre dos bignums
bool operator<(const bignum& a, const bignum& b)
{
  if(a.len<b.len)
    return true;
  else if(b.len<a.len)
    return false;
  else{
    for(int i=0; i<a.len; i++)
    {
      if(a.digits[i]<b.digits[i]){
        return true;
      }else if(b.digits[i]<a.digits[i])
        return false;
    }
  }
  return false;
}

//Sobrecarga del operador > para la comparacion entre dos bignums
bool operator>(const bignum& a, const bignum& b)
{
  if(a.len>b.len)
    return true;
  else if(b.len>a.len)
    return false;
  else{
    for(int i=0; i<a.len; i++)
    {
      if(a.digits[i]>b.digits[i]){
        return true;
      }else if(b.digits[i]>a.digits[i])
        return false;
    }
  }
  return false;
}
```

Listing 11: Sobrecarga de los operadores de comparación

## 3. Compilación

En cuanto al proceso de compilación se creó un archivo `Makefile` para obtener el ejecutable a partir de los archivos fuente:

```
PROGRAM=tp0
CC := g++
FLAGS := -Wall -pedantic -g
LDLFLAGS=-lm
GREEN= \e[92m
NORMAL= \e[0m

all:
	@echo "$(GREEN)Compilando ...$(NORMAL)"
	$(MAKE) tp0
	@echo "$(GREEN)Terminó$(NORMAL)"

cmdline.o: cmdline.cpp cmdline.h
	$(CC) $(FLAGS) cmdline.o

bignum.o: bignum.cpp bignum.h
	$(CC) $(FLAGS) bignumm.0

tp0: cmdline.cpp bignum.cpp tp0.cpp
	$(CC) $(FLAGS) $^ -o tp0

clean:
	@echo "$(GREEN)Limpiando ...$(NORMAL)"
	rm -vf *.o $(PROGRAM)
	@echo "$(GREEN)Listo!$(NORMAL)"

gdb: tp0
	gdb ./tp0

valgrind: tp0
	valgrind --leak-check=full --show-leak-kinds=all ./tp0
```

Listing 12: Archivo Makefile

De esta manera se simplifica el proceso de compilación ya que solo se necesita escribir el comando **make**. Otros comandos útiles son:

- **make clean:** borra todos los archivos .o generados.
- **make gdb:** para debuggear con gdb.
- **make valgrind:** ejecuta el programa con valgrind para verificar que no haya fallas de memoria.

```
luliberard@ThinkpadLuciaB:/mnt/c/Users/l_lul/Documents/Algoritmos2/TP0$ make clean
Limpiando ...
rm -vf *.o tp0
removed 'tp0'
Listo!
luliberard@ThinkpadLuciaB:/mnt/c/Users/l_lul/Documents/Algoritmos2/TP0$ make
Compilando ...
make tp0
make[1]: Entering directory '/mnt/c/Users/l_lul/Documents/Algoritmos2/TP0'
g++ -Wall -pedantic -g cmdline.cpp bignum.cpp tp0.cpp -o tp0
make[1]: Leaving directory '/mnt/c/Users/l_lul/Documents/Algoritmos2/TP0'
Terminó
luliberard@ThinkpadLuciaB:/mnt/c/Users/l_lul/Documents/Algoritmos2/TP0$ make gdb
gdb ./tp0
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 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-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./tp0...
(gdb) q
luliberard@ThinkpadLuciaB:/mnt/c/Users/l_lul/Documents/Algoritmos2/TP0$ make valgrind
valgrind --leak-check=full --show-leak-kinds=all ./tp0
==211== Memcheck, a memory error detector
==211== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==211== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==211== Command: ./tp0
```

Figura 3.1: Compilación utilizando los comandos del Makefile

## 4. Ejecución del programa

Luego de compilar el programa, se prosigue a ejecutar el mismo. Esto se realiza escribiendo **echo**, luego la operación que se quiera realizar, un pipeline, el archivo ejecutable **./tp0** y opcionalmente el comando de la precisión (-p) con su respectivo valor. Por ejemplo:

```
echo "-1 - 5" | ./tp0 -p 2
```

```
echo -189*50 | ./tp0 -p 6
```

La precisión ingresada afectará sólo a los valores de entrada con los que se operará, es decir, que el resultado podrá tener una precisión distinta de ser necesario, según la operación a realizar.

Otra de las opciones disponibles, es aquella que se ejecuta con el comando **-i** para realizar cuentas escritas en un archivo .txt aparte:

```
./tp0 -p 2 -i cuentas.txt
```

En el ejemplo, se realizarán las operaciones leídas del archivo *cuentas.txt*, las cuales deberán ocupar una línea cada una, de lo contrario se emitirá un error de operación.

Por otra parte, la opción correspondiente al comando **-o** puede utilizarse para abrir un archivo .txt donde se imprimirán los resultados de las cuentas ingresadas, por ejemplo:

```
./tp0 -p 2 -o resultados.txt
```

En el ejemplo, se imprimirán los resultados en el archivo *resultados.txt*, borrando el contenido previo de dicho archivo.

Se destaca, que en caso de no recibir el comando **-p**, la precisión del procesamiento es la necesaria para realizar la operación demandada, es decir, no se modificarán los valores ingresados para ser procesados. En el caso en que un número de entrada no pueda ser representado por la precisión indicada, se entenderá como precisión a la cantidad de cifras decimales significativas. Como ejemplo, si se ingresa el valor 12345 con una precisión de 4, la operación tomará el valor 12350. Es decir, las primeras 4 cifras significativas del número, redondeando la última cifra.

Por otra parte, si no se recibe el comando **-o**, se toma la salida estándar por defecto (cout). Análogamente con el comando **-i** (cin). Además, se aclara que en caso de que el archivo ingresado como argumento de éstas dos últimas opciones no exista, se expresa un mensaje de error de apertura del archivo.

A continuación, se observará la última opción disponible que se ingresa con el comando **-h**, el cual tiene como objetivo la impresión de un mensaje de ayuda que indica las opciones de ejecución posibles.

```
./tp0 -h
cmdline -p precision [-i file] [-o file]
```

## 5. Verificación con Valgrind y resultados de los casos de prueba

En primera instancia se ejecutó el programa con Valgrind para verificar que no haya fallas de memoria en el código. El programa espera que se ingrese operaciones a realizar y corta cuando se ingresa una operación inválida. Se obtuvo el siguiente mensaje:

```
luliberard@ThinkpadLuciaB:/mnt/c/Users/l_lul/Downloads/TP0-master$ make valgrind
valgrind --leak-check=full --show-leak-kinds=all ./tp0
==42== Memcheck, a memory error detector
==42== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==42== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==42== Command: ./tp0
==42==
==42== error calling PR_SET_PTRACER, vgdb might block
45*5
225
89+5
94
7995--5
8000
4
Operacion invalida
==42==
==42== HEAP SUMMARY:
==42==     in use at exit: 0 bytes in 0 blocks
==42==   total heap usage: 33 allocs, 33 frees, 81,220 bytes allocated
==42==
==42== All heap blocks were freed -- no leaks are possible
==42==
==42== For lists of detected and suppressed errors, rerun with: -s
==42== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
make: *** [Makefile:31: valgrind] Error 1
```

Figura 5.1: Verificación con Valgrind

Lo cual verifica el correcto funcionamiento del manejo de memoria a lo largo del código.

Luego se procedió a probar distintos casos para corroborar los resultados:

```
luliberard@ThinkpadLuciaB:/mnt/c/Users/l_lul/Documents/Algoritmos2/TP0$ echo 1586+8994 | ./tp0 -p 6
10580
luliberard@ThinkpadLuciaB:/mnt/c/Users/l_lul/Documents/Algoritmos2/TP0$ echo 1586*-4 | ./tp0 -p 6
-6344
luliberard@ThinkpadLuciaB:/mnt/c/Users/l_lul/Documents/Algoritmos2/TP0$ echo -1586*-4 | ./tp0 -p 6
6344
luliberard@ThinkpadLuciaB:/mnt/c/Users/l_lul/Documents/Algoritmos2/TP0$ echo -15860000-40000000045748 | ./tp0 -p 20
-40000015905748
luliberard@ThinkpadLuciaB:/mnt/c/Users/l_lul/Documents/Algoritmos2/TP0$ echo -15860000*40000000045748 | ./tp0 -p 20
-634400000725563280000
luliberard@ThinkpadLuciaB:/mnt/c/Users/l_lul/Documents/Algoritmos2/TP0$ echo 895*0 | ./tp0 -p 20
0
luliberard@ThinkpadLuciaB:/mnt/c/Users/l_lul/Documents/Algoritmos2/TP0$ echo 895*-1 | ./tp0 -p 20
-895
luliberard@ThinkpadLuciaB:/mnt/c/Users/l_lul/Documents/Algoritmos2/TP0$ ./tp0 -i cuentas.txt
18
12
28
luliberard@ThinkpadLuciaB:/mnt/c/Users/l_lul/Documents/Algoritmos2/TP0$ echo  | ./tp0 -p 5
luliberard@ThinkpadLuciaB:/mnt/c/Users/l_lul/Documents/Algoritmos2/TP0$ echo error | ./tp0 -p 5
Operacion invalida
luliberard@ThinkpadLuciaB:/mnt/c/Users/l_lul/Documents/Algoritmos2/TP0$ echo 89+54+5 | ./tp0 -p 5
Operacion invalida
luliberard@ThinkpadLuciaB:/mnt/c/Users/l_lul/Documents/Algoritmos2/TP0$ echo 1+     5 | ./tp0 -p 5
6
luliberard@ThinkpadLuciaB:/mnt/c/Users/l_lul/Documents/Algoritmos2/TP0$ echo 1+     *5 | ./tp0 -p 5
Operacion invalida
luliberard@ThinkpadLuciaB:/mnt/c/Users/l_lul/Documents/Algoritmos2/TP0$ echo -    5  +895 | ./tp0 -p 5
890
luliberard@ThinkpadLuciaB:/mnt/c/Users/l_lul/Documents/Algoritmos2/TP0$ echo 985540+55445 | ./tp0 -p 1
1060000
luliberard@ThinkpadLuciaB:/mnt/c/Users/l_lul/Documents/Algoritmos2/TP0$ echo 29+58 | ./tp0 -p 1
90
```

Figura 5.2: Casos de prueba

```
diego_sando@LAPTOP-TJB1CSV1:/mnt/c/users/user/desktop/fiuba/algoritmos2/ejercicios/tp0$ make
Compilando ...
make tp0
make[1]: Entering directory '/mnt/c/users/user/desktop/fiuba/algoritmos2/ejercicios/tp0'
g++ -Wall -pedantic -g cmdline.cpp bignum.cpp tp0.cpp -o tp0
make[1]: Leaving directory '/mnt/c/users/user/desktop/fiuba/algoritmos2/ejercicios/tp0'
Termino
diego_sando@LAPTOP-TJB1CSV1:/mnt/c/users/user/desktop/fiuba/algoritmos2/ejercicios/tp0$ ./tp0 -i cuentas.txt -o resultados.txt
diego_sando@LAPTOP-TJB1CSV1:/mnt/c/users/user/desktop/fiuba/algoritmos2/ejercicios/tp0$
```

cuentas: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
1+ 1
11 *2
9999999999 + 1
111 - -4
-1111 + -5
+999    -   111
15*3
```

resultados: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
2
22
10000000000
115
-1116
888
45
```

Figura 5.3: Casos de prueba con archivos .txt

## 6. Conclusiones

A lo largo del trabajo práctico se logró diseñar e implementar una clase bignum que representa números enteros de precisión fija pero arbitrariamente grande. Esto sirvió para comprender mejor el funcionamiento interno al hacer operaciones con distintos tipos de datos y la complejidad que eso conlleva. A diferencia del lenguaje de programación C, donde para generar este tipo de operaciones se debe armar funciones especificas (por ejemplo "sumar(a,b)"), en este trabajo se utilizó la ventaja de C++ de poder sobrecargar los operadores para que también puedan realizar operaciones con la clase bignum definida de una manera más intuitiva (a+b).

A la hora de dividir las tareas del trabajo, se tuvo que tener especial cuidado que todos los módulos y funciones dentro de la clase bignum sean compatibles con el desarrollo de los demas integrantes. Es por eso que se tuvo que definir de antemano cómo se iba a encarar el proyecto. Se trato de diseñar funciones compactas y eficientes pero que a la vez mantuvieran un código legible, facilitando el trabajo en grupo. Se trabajó utilizando la plataforma Github para trabajar siempre bajo las últimas modificaciones.

## 7. Anexo - Códigos

Se adjunta los códigos utilizados en el siguiente link de Google Drive:

https://drive.google.com/drive/folders/1rLj7F6r3_dt0o9U1ecpbRClZs7u-khmJ?usp=sharing