75.04/95.12 Algoritmos y Programación II
Trabajo Práctico 0: Programación C++

Universidad de Buenos Aires - FIUBA
Primer cuatrimestre de 2021

Alumnos:

- Clavijo, Jorge - 95485 - joclavijo@fi.uba.ar
- Lo Coco, Manuel - 103938 - mlococo@fi.uba.ar
- Mesquida, Christian - 96148 - cmesquida@fi.uba.ar
- Strangis, Nicolás - 96407 - nstrangis@fi.uba.ar

Fecha de 1er entrega: 27/05/2021

FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Enunciado

1. Objetivos

Ejercitar los conceptos básicos de programación C++ vistos en las primeras clases. Familiarizarse con algunas de las herramientas de software que usamos en el curso y en los siguientes trabajos prácticos, implementando un programa (y su correspondiente documentación) que resuelva el problema que presentaremos más abajo.

2. Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

3. Requisitos

El trabajo deberá ser entregado personalmente, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes, un informe impreso de acuerdo con lo que mencionaremos en la seccion 5, y con una copia digital de los archivos fuente necesarios para compilar el trabajo.

4. Descripción

El objetivo fundamental del trabajo es implementar números enteros de precisión fija (pero arbitrariamente grande). Entederemos como precisión a la cantidad de dígitos decimales que conforman el número.
En una primera etapa, deseamos representar este concepto utilizando arreglos. Para ello, desarrollaremos 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, será preciso además sobrecargar los operadores aritméticos de suma, resta y multiplicación y los de entrada y salida con formato. Esto se debe a que, a los fines de introducirse en el lenguaje y familiarizarse con sus construcciones, realizaremos un ejercicio sencillo que consiste en tomar expresiones aritméticas de dos operandos desde streams configurables y resolverlas, informando el resultado en otro stream de salida, también configurable.

Las porciones marcadas con puntos suspensivos corresponden a los restantes métodos y/o variables necesarias para el correcto funcionamiento del programa. Su elección queda a criterio de cada grupo.

Especificaremos tanto los nombres de los streams como así también la precisión con la que trabajaremos a través de opciones de línea de comando (ver sección 4.2).

4.1. Consideraciones algorítmicas

Si bien existen diversos algoritmos para resolver el problema de la multiplicación de números de precisión arbitraria, en este primer acercamiento la idea será implementar el algoritmo trivial de multiplicación enseñado en la escuela primaria. En instancias de evaluación posteriores retomaremos esta problemática e implementaremos algoritmos de mayor sofisticación, dejando también espacio para un análisis riguroso que nos permita comparar ambas alternativas.

4.2. Línea de comando

Las opciones -i y -o permiten seleccionar los streams de entrada y salida respectivamente. Por defecto, estos serán cin y cout. Lo mismo ocurrirá al recibir "-" como argumento.

Por otro lado, la opción -p indicará el valor de la precisión con la que llevaremos 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. Cada grupo deberá definir y documentar en forma adecuada que hacer en caso de recibir un número no representable con la precisión definida.

Al finalizar, todos nuestros programas retornarán un valor nulo en caso de no detectar ningún problema; en caso contrario, devolveremos un valor no nulo (por ejemplo 1).

4.3. 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.

A continuación, presentaremos algunos ejemplos puntuales de estos archivos.

4.4. Ejemplos

Primero, usamos un archivo vacío como entrada del programa:

$ ./tp0 -i /dev/null

Notar que la salida es también vacía. En el siguiente ejemplo, pasamos una única operación como entrada. La salida deberá contener sólo una línea con el resultado adecuado:

$ echo 1123581321345589*123456789 | ./tp0 -p 20

138713742113703577253721

No olvidemos que los números son enteros: también pueden ser negativos. El siguiente ejemplo muestra un correcto comportamiento del programa frente a estos casos:

$ echo 1 - 5"| ./tp0 -p 1

-6

Notar además que la precisión puede ser incluso mayor a la cantidad de dígitos del número de mayor valor absoluto contenido en la entrada.

4.5. Portabilidad

A los efectos de este primer trabajo, es deseable (pero no requisito) que la implementación desarrollada provea un grado mínimo de portabilidad. Sugerimos verificar nuestros programas en Windows/MS-DOS y/o alguna versión reciente de UNIX: BSD, o Linux

5. Informe

El informe deberá incluir:

- Una carátula que incluya los nombres de los integrantes y el listado de todas las entregas realizadas hasta ese momento, con sus respectivas fechas
- Documentación relevante al diseño e implementación del programa.
- Documentación relevante al proceso de compilación: cómo obtener el ejecutable a partir de los archivos fuente.
- Las corridas de prueba, con los comentarios pertinentes.
- El código fuente, en lenguaje C++ (en dos formatos, digital e impreso).
- Este enunciado.

6. Fechas

La última fecha de entrega y presentación será el jueves 27 de mayo.

Diseño e implementación del programa:

Desarrollamos nuestro trabajo en base a la clase **bignum** preestablecida en el ítem 4. Agregamos los atributos y métodos que consideramos necesarios para resolver el problema planteado, quedando la clase conformada de la siguiente manera:

```
class bignum
{
	private:
		unsigned short *digits;
		int dim;
		bool sign;
	public:
		bignum();
		bignum(const bignum&);
		bignum(std::string&, int);
		bignum(int);
		 ~ bignum();

		friend bignum operator+(const bignum&, const bignum&);
		friend bignum operator-(const bignum&, const bignum&);
		friend bignum operator*(const bignum&, const bignum&);
		friend bignum operator*(const bignum& a, const unsigned short b);
		friend std::ostream& operator«(std::ostream&, const bignum&);
		friend std::istream& operator»(std::istream&, bignum&);
};
```

Una vez codificada la información ingresada por el usuario (ampliaremos esta sección más adelante), se creará una instancia de bignum mediante el constructor $bignum(std :: string\&, int)$; el cual recibe por argumentos un *string* (contiene uno de los números de la operación) y un *int* (indica la precisión).

En lo que respecta a los atributos, *digits* es un vector dinámico que almacena en cada celda un dígito del número, y lo hace al revés de la convención normal utilizada, el dígito menos significativo se almacena en la posición cero del vector y el más significativo en la última posición de éste.

En la variable *dim* se guarda la dimensión del vector, ya que nos será de extrema utilidad a la hora de recorrerlo. Con el último atributo booleano *sign* identificaremos

el signo, $false$ para números positivos y $true$ para negativos (por defecto es $false$).

En cuanto a los constructores, $bignum()$ creará una instancia "vacía"de bignum, cuyo $digits$ es NULL y su dimensión cero. Tenemos el constructor por copia $bignum(const\ bignum\&)$, $bignum(int)$ que recibe un entero y crea un vector con ese valor, inicializado con ceros y $bignum(std :: string\&, int)$ mencionado anteriormente. Además definimos el destructor $\sim bignum()$ que libera la memoria utilizada.

Por último tenemos las sobrecargas de los operadores que utilizaremos en las operaciones entre dos bignum ($+$, $-$, $*$) y los operadores de entrada y salida ($\gg$ y $\ll$). En todos los casos explicaremos su funcionamiento ya que son una instancia fundamental del código.

Sobrecarga del operador '+':

En el operador suma, en primer lugar se verifica que los sumandos sean del mismo signo. En caso de que no sean del mismo signo se llama al operador resta y se devuelve el resultado obtenido. Si son del mismo signo se crea un bignum c, mediante el constructor $bignum(int)$. La dimensión de c será la mayor entre a y b, sumado 1 por si ocurre un carry en la última iteración. Luego, se crean 2 vectores aa y bb con la función auxiliar $copy\_array$. Dicha función crea una copia de $digits$ en un vector auxiliar de dimensión dim. En $c.digits[i]$ se va almacenando el resultado de $(aa[i] + bb[i] + carry_i)\,\%10$. Siendo $carry_i$ el resultado de de $(aa[i-1] + bb[i-1] + carry_{i-1})/10$. Para el caso de $i = 0$ se toma que el carry vale 0. Finalmente, una vez que se terminó la iteración, se eliminan los vectores auxiliares aa y bb, se llama a la función $resize$ para eliminar los ceros sobrantes de c y se devuelve c.

Sobrecarga del operador '−':

Dentro del desarrollo de este operador, se implementan algunas funciones auxiliares. Éstas se definieron como estáticas ya que son utilizadas de forma local por el operador resta, sin tener una relación directa con la clase. Por lo tanto, se mantiene su ámbito privado, sin acceso para el usuario.

La primer etapa de la resta consiste en una validación según el módulo de los números mediante la función $modulo\_igual$, si tenemos una operación $c = a - b$ y $|a| = |b|$, sean $a, b > 0$ ó $a, b < 0$ el resultado será $c = 0$. Este análisis lo podemos interpretar lógicamente como una compuerta XNOR. En caso de ser $|a| = |b|$ pero diferir en su

signo, tendremos dos opciones, $c = a + a$ o $c = -(a + a)$. Como se observa, interactuamos con un operador dentro de otro.

Cuando $|a| \neq |b|$, si $a < 0$ y $b > 0$ tendremos $c = -(a + b)$, y si $a > 0$ y $b < 0$ será $c = a + b$, ambos resultados del operador suma. Por último, consideramos los casos en que difieren su módulo pero tienen el mismo signo, para los cuáles verificamos con la función *mayor* si $b > a$ se evaluará la función *resta* como $c = -(b - a)$. Caso contrario si $a > b$, $c = a - b$.

La función *resta* recibe como parámetros los *digits* de las dos instancias con sus respectivos tamaños y una variable como referencia en la que establece la dimensión del vector resultante, al cuál lo retorna por valor. Dentro de esta función se realiza el proceso algorítmico básico de la resta, en el cuál se resta independientemente dígito por dígito y se toma una unidad del dígito siguiente en caso de ser necesario.

Eventualmente, este proceso requerirá disminuir indefinidamente la dimensión del resultado respecto a los dos operandos, por lo que se implementa la función *resize*, que elimina los ceros redundantes del vector resultante (por interfaz) y devuelve su nueva dimensión por valor. Al termino de esta operación, se verifica el signo de la resta y se crea una nueva instancia de bignum con los datos obtenidos, que se retorna como un nuevo objeto, sin alterar a ni b. Adicionalmente, para realizar la copia de vectores se utiliza la función *copy_array*, que simplemente copia un vector *origen* en uno *destino*.

Sobrecarga del operador '$*$':

Se realizaron dos sobrecargas del operador '$*$', una para cuando se reciben dos objetos *bignum* y una para cuando se recibe un objeto *bignum* y un dígito *unsigned short*, ésta ultima se realizó para ser usada por la primera. Luego, además, dadas las características de la multiplicación fue necesario un método para agregar ceros "detrás"de los dígitos menos significativos de una instancia de objeto.

Lo primero que se realiza en la sobrecarga es, mediante el constructor *bignum*(*int*), construir un objeto que tenga de largo la suma de las dimensiones de los objetos recibidos. Luego se itera a través de uno de los objetos accediendo a cada uno de sus dígitos y multiplicándolo utilizando la otra sobrecarga del operador guardando el resultado en otra instancia de objeto creada en la iteración. Luego se llama al método *agregar_ceros* que funciona como un corrimiento para así por fin, a través del operador suma guardarlo en el objeto que será devuelto al final de la sobrecarga.

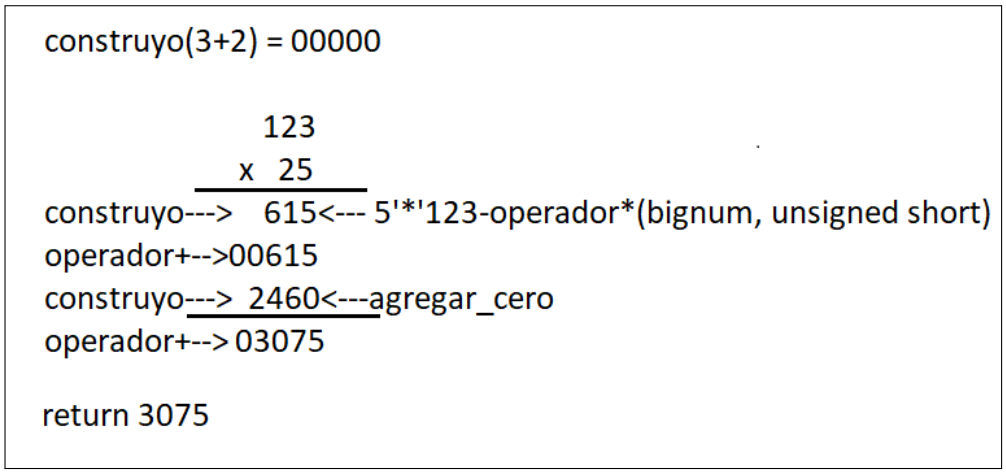

Figura 1: funcionamiento de la sobrecarga

Para definir el signo de la devolución se verifica si los signos de los objetos entrantes son iguales. Si son iguales se retornara un valor positivo y caso contrario un valor negativo.

Sobrecarga del operador '=':

Para la sobrecarga de este operador, se asume para la explicación que se realiza la operación $a = b$, siendo a y b dos bignum. Cuando llamamos $a = b$ es lo mismo que realizar la operación $a.operator = (b)$, por ende a es el objeto en el que estamos trabajando y b es el argumento. En primer lugar se verifica si las referencias son iguales, en caso de serlo se devuelve el objeto en el que se está trabajando(lo devolvemos al llamar $*this$). Luego, se verifican ambas dimensiones, si las dimensiones son distintas, se crea un nuevo vector con la dimensión de b, se le asigna a a el mismo signo y la misma dimensión que tiene b, se elimina $digits$ y se llama a la función $copy\_array$ para copiar los elementos de b en a. Luego, se devuelve $*this$. En caso de que las dimensiones sean distintas, se le asigna a a el mismo signo que tiene b se llama a la función $copy\_array$ y se devuelve $*this$.

Streams de Entrada/Salida y Validaciones:

Se implementa el código sugerido en clase, el cual, consiste en el manejo de los flujos de entrada y salida mediante las clases estándar **istream** y **ostream**. La entrada estándar habitual es el objeto *cin*, el cual es un objeto de tipo *istream*. De igual manera los objetos estándar *clog*, *cout*, y *cerr* son objetos de tipo *ostream*, en el trabajo práctico se usan los dos últimos.

También se tiene la clase estándar *fstream* la cual realiza el manejo de streams de entrada y salida para operar con archivos

Dentro de los argumentos de entrada, la opción precisión **-p** se almacena en la variable global *static precision_t precision*, si el usuario omite la precisión se toma como su valor a todo el número ingresado, es decir, precisión total para la operación.

Una vez que el usuario define los argumentos de entrada para el manejo de entrada/-salida, estos flujos se encuentran almacenados en las variables globales *static istream *iss*, y *static ostream *oss*.

La clase *precision_fija* recibe los flujos por referencia al construirse una instancia de la clase y los procesa en el método *precision_fija::captura(precision_t *precision)*, en este método es en donde confluyen las distintas clases en el programa y se realiza todo el proceso de operaciones aritméticas.

En el método de captura al flujo de entrada se lo procesa línea por línea. Para que una línea se válida tendrá que pasar un procesamiento de **regex**, se usa la clase *std::basic_regex* para este propósito, la expresión regular implementada consta de 3 grupos de captura, el primero y tercero valida un número con signo o sin él y el segundo grupo de captura valida la operación que se va a realizar.

Entre el primer y el segundo grupo de captura, al igual que entre el segundo y tercer grupo de captura se válida los caracteres especiales \s que es equivalente a \r, \n, \t, \f, \v.

El programa termina al encontrar un EOF en el flujo de entrada o al procesar una línea vacía, en este caso se enviará al flujo de salida *Finished program*, el programa devuelve 0 a menos que haya habido un error.

En el caso de que una línea no pase el regex, se envía al flujo de salida el mensaje *Entry not processed* y se continua con el programa, en este caso el programa devolverá un 1 a pesar de que las siguientes operaciones sean operaciones válidas.

No se sobrecarga el operador de entrada ya que los **bignum** en cada operación de cada línea son creados mediante constructor que toma un **string** válido y la precisión elegida. Para la implementación con listas template se sobrecargará este operador.

El operador de salida se encuentra sobrecargado y le envía al flujo de salida el signo y los dígitos de la operación resultante.

Compilación del Programa:

Creamos un archivo Makefile para compilar conjuntamente todo el proyecto. Consiste en:

```
CC = g++
CFLAGS = -Wall -g


tp0: bignum.o cmdline.o precisionfija.o tp0.o
	$(CC) $(CFLAGS) -o tp0 bignum.o cmdline.o precisionfija.o tp0.o

tp0.o: tp0.cpp bignum.h cmdline.h precisionfija.h
	$(CC) $(CFLAGS) -c tp0.cpp cmdline.o: cmdline.h
	$(CC) $(CFLAGS) -c cmdline.cpp precisionfija.o: precisionfija.h
	$(CC) $(CFLAGS) -c precisionfija.cpp

bignum.o: bignum.h
	$(CC) $(CFLAGS) -c bignum.cpp

clean:
	@rm -f *.o *.out *.in tp0
```

Para compilarlo se ejecuta $ *make* y para eliminar los ejecutables $ *make clean*

Pruebas:

Para la primera prueba se usa un programa en python **randop.py** el cual al ejecutarse crea dos archivos: big.in (las operaciones) y big.out (los resultados). Por defecto el número máximo de operaciones que genera el programa son 1000 con precisión mínima 20 y precisión máxima 100.

Se usa la utilidad de comparación de archivos **diff** para testear el programa. El resultado es el siguiente:

```
→  AlgoritmosII git:(development) make
g++ -Wall -g -c bignum.cpp
g++ -Wall -g -c cmdline.cpp
g++ -Wall -g -c precisionfija.cpp
g++ -Wall -g -c tp0.cpp
g++ -Wall -g -o tp0 bignum.o cmdline.o precisionfija.o tp0.o
→  AlgoritmosII git:(development) ✗ python3.9 randop.py
→  AlgoritmosII git:(development) ✗ ./tp0 -i big.in -o big_tp0.out
→  AlgoritmosII git:(development) ✗ diff big.out big_tp0.out
→  AlgoritmosII git:(development) ✗ echo $?
0
→  AlgoritmosII git:(development) ✗
```

La utilidad **diff** y el programa terminan correctamente.

Todos los ejemplos de prueba señalados en la sección 4.4 del informe se ejecutan adecuadamente.

```
→  AlgoritmosII git:(development) ✗ ./tp0 -i /dev/null
→  AlgoritmosII git:(development) ✗ echo "1123581321345589*123456789" | ./tp0 -p 20
138713742113703577253721
→  AlgoritmosII git:(development) ✗ echo "-1      -    5" | ./tp0 -p 1
-6
→  AlgoritmosII git:(development) ✗
```

Se evidencia la opción **-i -** para la entrada estándar, también como afecta la precisión a la entrada en la multiplicación, en la tercera línea al ser una operación no válida el programa informa con un mensaje de error y continua su operación hasta encontrar una línea vacía, sin embargo, ya que hubo un error el programa devuelve **1**.

```
→  AlgoritmosII git:(development) ✗ ./tp0 -i - -p 5 -o big.out
515154151*515151
54654a*4
10000+1

→  AlgoritmosII git:(development) ✗ echo $?
1
→  AlgoritmosII git:(development) ✗ cat big.out
265379522500000
Entry not processed
10001
Finished program
→  AlgoritmosII git:(development) ✗
```

Finalmente se chequea la **perdida de memoria**, para este propósito se usa el programa **valgrind** con la opción *–leak-check=full* que muestra cada perdida de memoria en detalle y una lista de errores si los hubiese usando la opción *-s* que es la versión corta de *-show-error-list=yes*.

```
→  AlgoritmosII git:(development) ✗ python3.9 randop.py
→  AlgoritmosII git:(development) ✗ wc -l big.in
449 big.in
→  AlgoritmosII git:(development) ✗ valgrind -s --leak-check=full ./tp0 -i big.in -o big_tp0.out
==107824== Memcheck, a memory error detector
==107824== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==107824== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==107824== Command: ./tp0 -i big.in -o big_tp0.out
==107824==
==107824==
==107824== HEAP SUMMARY:
==107824==     in use at exit: 0 bytes in 0 blocks
==107824==   total heap usage: 98,421 allocs, 98,421 frees, 17,060,852 bytes allocated
==107824==
==107824== All heap blocks were freed -- no leaks are possible
==107824==
==107824== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
→  AlgoritmosII git:(development) ✗
```

Se procesan 449 operaciones y el resultado muestra que no existe forma de que se haya filtrado memoria durante la ejecución del programa.

Referencias:

[1] Wikipedia, “Long multiplication.” https://en.wikipedia.org/wiki/Multiplication_algorithmLong_multiplication

[2] https://en.cppreference.com/w/

[3] “C++ How to Program - Paul Deitel, Harvey Deitel”

Código

bignum.h

```
#ifndef _BIGNUM_H_INCLUDED_
#define _BIGNUM_H_INCLUDED_

#include <iostream>
#include <sstream>
#include <string>
#include <stdio.h>

using namespace std;
class bignum
{
private:
        unsigned short *digits;
        int dim;
        bool sign;

public:
        bignum ();
        bignum (const bignum&);
        bignum (std::string&, int);
        bignum (int);

        ~bignum();

        bignum agregar_ceros(int pos, int n);
        bignum& operator=(const bignum&);

        friend bignum operator*(const bignum& a, const unsigned short b);
        friend bignum operator*(const bignum& a, const bignum& b);
        friend bignum operator+(const bignum& a, const bignum& b);
        friend bignum operator-(const bignum& a, const bignum& b);

        bignum convertir_bignum(std::string&);
        void emitir_bignum();
        friend std::ostream& operator<<(std::ostream&, const bignum&);
        friend std::istream& operator>>(std::istream&, bignum&);
};
#endif
```

bignum.cpp

```
#include "bignum.h"

bignum::bignum()
{
    digits = NULL;
    dim = 0;
    sign = false;
}
bignum::bignum(int n)
{
    digits = new unsigned short[n]();
    dim = n;
    sign = false;
}
bignum::bignum(const bignum &a)
{
    digits = new unsigned short[a.dim];
    for(int i = 0; i < a.dim; i++)
        digits[i] = a.digits[i];
    dim = a.dim;
    sign = a.sign;
}
bignum::bignum(std::string& str, int precision)
{
    dim = str.length();
    sign = false;
    int j = 0; //Contador del string
    if(str.at(0) == '-')
    {
        sign = true;
        precision++; //Contemplo el '-' en la precision
        dim--;
        j++;
    }
    digits = new unsigned short[dim];
    for(int i = dim - 1; i >= 0; i--)
    {
        if(j < precision)
        {
            char c = str.at(j++);
            digits[i] = c - '0';
        }
        else
            digits[i] = 0;
    }
}
bignum::~bignum()
{
    if(digits)
```

```
        delete [] digits;
}

void bignum::emitir_bignum()
{
for(int i = 0; i < dim; i++)
    cout << digits[i];

cout << ", " << (sign ? "NEGATIVO" : "POSITIVO") << endl;
}
static void copy_array(unsigned short *dest, unsigned short *orig, int n)
{
    for(int i = 0; i < n; i++)
        dest[i] = orig[i];
}

static int resize(unsigned short *&a, int n)
{
    int ceros = 0;
    while((a[n - ceros - 1] == 0) && (n - ceros - 1) > 0) ceros++;
    unsigned short *aux = new unsigned short[n - ceros];
    copy_array(aux, a, n - ceros);
    delete[] a;
    a = aux;
    return n - ceros;
}

bignum bignum::agregar_ceros(int pos, int n)
{
    unsigned short *aux = new unsigned short[dim + n]();
    copy_array(aux + n, digits, dim);
    delete [] digits;
    digits = aux;
    dim += n;
    return *this;
}

bignum& bignum::operator=(const bignum& b)
{
    if(&b != this)
    {
        if(dim != b.dim)
        {
            unsigned short *aux = new unsigned short[b.dim];
            delete[] digits;
            dim = b.dim;
            sign = b.sign;
            digits = aux;
            copy_array(digits, b.digits, dim);
            return *this;
        }
```

```
        else
        {
            sign = b.sign;
            copy_array(digits, b.digits, dim);
            return *this;
        }
    }
    return *this;
}

bignum operator*(const bignum& a, const bignum& b)
{
    int largo = a.dim + b.dim;

    bignum retorno(largo);
    for (int k = 0; k < b.dim; k++)
    {
        bignum multi(a.dim + 2 + k);
        multi = a * b.digits[k];
        multi.sign = false;
        retorno.sign = false;
        retorno = retorno + multi.agregar_ceros(a.dim + 1, k);
    }
    b.sign == a.sign ? retorno.sign = false : retorno.sign = true;
    return retorno;
}

bignum operator*(const bignum& a, const unsigned short b)
{
    bignum resultado(a.dim + 1);
    int i = 0;
    unsigned short carry = 0;
    for (; i < a.dim; )
    {
        unsigned short multi = 0;
        multi = a.digits[i] * b;
        if(multi + carry > 9)
            resultado.digits[i] = (multi + carry) %10;
        else
            resultado.digits[i] = multi + carry;
        resultado.digits[i + 1] = (multi + carry) / 10;
        carry = 0;
        carry = resultado.digits[i + 1];
        i++;
    }
    return resultado;
}

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

```
    a.dim > b.dim ? new_dim = a.dim + 1 : new_dim = b.dim + 1;
    bignum c(new_dim);
    if(a.sign && !b.sign) // a < 0 y b > 0 --> c = b - a
    {
        bignum aa(a);
        aa.sign = false;
        c = b - aa;
        return c;
    }
    if(!a.sign && b.sign) // a > 0 y b < 0 --> c = a - b
    {
        bignum bb(b);
        bb.sign = false;
        c = a - bb;
        return c;
    }
    unsigned short *aa = new unsigned short[new_dim]();
    unsigned short *bb = new unsigned short[new_dim]();
    copy_array(aa, a.digits, a.dim);
    copy_array(bb, b.digits, b.dim);
    c.sign = a.sign;
    for(int i = 0; i < new_dim; i++)
    {
        unsigned short carry = 0;
        unsigned short suma = 0;
        suma = aa[i] + bb[i] + c.digits[i];
        carry = suma / 10;
        c.digits[i] = suma %10;
        if(i < new_dim - 1)
            c.digits[i + 1] = carry;
    }
    c.dim = resize(c.digits, new_dim);
    delete []aa;
    delete []bb;
    return c;
}

static bool mayor(unsigned short *v1, size_t n1, unsigned short *v2,
                  size_t n2)
{
    if(n1 > n2)
        return true;
    if(n1 < n2)
        return false;
    else {
        size_t i = n1 - 1;
        while(v1[i] == v2[i]) i--;
        if(v1[i] > v2[i]) return true;
        return false;
    }
}
```

```
static bool modulo_igual(unsigned short *v1, int n1, unsigned short *v2,
                         int n2)
{
    if(n1 != n2) return false;

    for(int i = 0; i < n1; i++)
        if(v1[i] != v2[i]) return false;
    return true;
}
static unsigned short *resta(unsigned short *a, int na, unsigned short *b,
                             int nb, int &nc)
{
    unsigned short *c = new unsigned short[na]();
    nc = na;

    for(int i = 0; i < nb; i++)
    {
        if(a[i] < b[i])              //Pide carry
        {
            a[i] += 10;
            c[i] = a[i] - b[i];
            if(a[i + 1] != 0)        //Si el que sigue no es cero le resta 1
                a[i + 1]--;
            else                 //Si es cero, sigue hasta encontrar un num > 0
            {
                int j = i + 1;
                while(a[j] == 0)
                {
                    a[j] = 9;
                    j++;
                }
                a[j]--;
            }
        }
        else if(a[i] > b[i])
            c[i] = a[i] - b[i];
        else c[i] = 0;
    }
    for(int i = nb; i < na; i++)     //Completa los restantes
        c[i] = a[i];

    if(c[na - 1] == 0) //Si cambia la dimension de la resta, la ajusta
        nc = resize(c, na);
    return c;
}
bignum operator-(const bignum& a, const bignum& b)
{
    bignum c; // c.digits = {0};
    int dim_c;

    unsigned short *aux;
```

```
    if(modulo_igual(b.digits, b.dim, a.digits, a.dim))
    {
        if(!(b.sign ^ a.sign)) return c;
        c = a + a;
        if(b.sign)
            return c;       // b < 0 y a = b --> c = a + a
        c.sign = true;              // a < 0 y a = b --> c = -(a + a)
        return c;
    }
    if(a.sign && !b.sign)           // a < 0 y b > 0 --> c = -(a + b)
    {
        bignum aa(a);
        aa.sign = false;
        c = aa + b;
        c.sign = true;
        return c;
    }
    if(!a.sign && b.sign)           // a > 0 y b < 0 --> c = a + b
    {
        bignum bb(b);
        bb.sign = false;
        c = a + bb;
        return c;
    }
    if(mayor(b.digits, b.dim, a.digits, a.dim)) //  b > a --> c = -(b - a)
    {
        aux = resta(b.digits, b.dim, a.digits, a.dim, dim_c);
        c.digits = new unsigned short[dim_c];
        c.dim = dim_c;
        if(!a.sign)                         // b > a, a > 0 y b > 0 --> c < 0
            c.sign = true;
        copy_array(c.digits, aux, dim_c);
        delete []aux;
        return c;
    }                                                   // a > b --> c = a - b
    aux = resta(a.digits, a.dim, b.digits, b.dim, dim_c);
    c.digits = new unsigned short[dim_c];
    c.dim = dim_c;
    if(a.sign)                      // a > b, a < 0 y b < 0 --> c < 0
        c.sign = true;
    copy_array(c.digits, aux, dim_c);
    delete []aux;
    return c;
}
std::ostream& operator<<(std::ostream& oss_, const bignum& out){
        if(out.dim == 0){
        oss_<<'0';
    }
    else{
        oss_<< (out.sign ? "-" : "");
        for(int i=out.dim; i!=0; i--){
```

```
                    oss_<< out.digits[i-1];
            }
    }
    oss_<<"\n";
    return oss_;
}
```

cmdline.h

```
#ifndef _CMDLINE_H_INCLUDED_
#define _CMDLINE_H_INCLUDED_

#include <cstdlib>
#include <string>
#include <iostream>

#define OPT_DEFAULT   0
#define OPT_SEEN      1
#define OPT_MANDATORY 2

struct option_t {
    int has_arg;
    const char *short_name;
    const char *long_name;
    const char *def_value;
    void (*parse)(std::string const &);
    int flags;
};


class cmdline {
    // Este atributo apunta a la tabla que describe todas
    // las opciones a procesar. Por el momento, solo puede
    // ser modificado mediante contructor, y debe finalizar
    // con un elemento nulo.
    //
    option_t *option_table;

    // El constructor por defecto cmdline::cmdline(), es
    // privado, para evitar construir parsers sin opciones.
    //
    cmdline();
    int do_long_opt(const char *, const char *);
    int do_short_opt(const char *, const char *);
public:
    cmdline(option_t *);
    void parse(int, char * const []);
};

#endif
```

cmdline.cpp

```
#include "cmdline.h"

using namespace std;

cmdline::cmdline()
{
}

cmdline::cmdline(option_t *table) : option_table(table)
{
}

void
cmdline::parse(int argc, char * const argv[])
{
#define END_OF_OPTIONS(p)    \
    ((p)->short_name == 0    \
     && (p)->long_name == 0 \
     && (p)->parse == 0)

    // Primer pasada por la secuencia de opciones: marcamos
    // todas las opciones, como no procesadas. Ver codigo de
    // abajo.
    //
    for (option_t *op = option_table; !END_OF_OPTIONS(op); ++op)
        op->flags &= ~OPT_SEEN;

    // Recorremos el arreglo argv. En cada paso, vemos
    // si se trata de una opcion corta, o larga. Luego,
    // llamamos a la funcion de parseo correspondiente.
    //
    for (int i = 1; i < argc; ++i) {
        // Todos los parametros de este programa deben
        // pasarse en forma de opciones. Encontrar un
        // parametro no-opcion es un error.
        //
        if (argv[i][0] != '-') {
            cerr << "Invalid non-option argument: "
            << argv[i]
            << endl;
            exit(1);
        }

        // Usamos "--" para marcar el fin de las
        // opciones; todo los argumentos que puedan
        // estar a continuacion no son interpretados
        // como opciones.
```

```
        //
        if (argv[i][1] == '-'
            && argv[i][2] == 0)
            break;

        // Finalmente, vemos si se trata o no de una
        // opcin larga; y llamamos al metodo que se
        // encarga de cada caso.
        //
        if (argv[i][1] == '-')
            i += do_long_opt(&argv[i][2], argv[i + 1]);
        else
            i += do_short_opt(&argv[i][1], argv[i + 1]);
    }

    // Segunda pasada: procesamos aquellas opciones que,
    // (1) no hayan figurado explicitamente en la linea
    // de comandos, y (2) tengan valor por defecto.
    //
    for (option_t *op = option_table; !END_OF_OPTIONS(op); ++op) {
#define OPTION_NAME(op) \
        (op->short_name ? op->short_name : op->long_name)
        if (op->flags & OPT_SEEN)
            continue;
        if (op->flags & OPT_MANDATORY) {
            cerr << "Option "
            << "-"
            << OPTION_NAME(op)
            << " is mandatory."
            << "\n";
            exit(1);
        }
        if (op->def_value == 0)
            continue;
        op->parse(string(op->def_value));
    }
}

int
cmdline::do_long_opt(const char *opt, const char *arg)
{
    // Recorremos la tabla de opciones, y buscamos la
    // entrada larga que se corresponda con la opcion de
    // linea de comandos. De no encontrarse, indicamos el
    // error.
    //
    for (option_t *op = option_table; op->long_name != 0; ++op) {
        if (string(opt) == string(op->long_name)) {
            // Marcamos esta opcion como usada en
            // forma explicita, para evitar tener
            // que inicializarla con el valor por
```

```
            // defecto.
            //
            op->flags |= OPT_SEEN;

            if (op->has_arg) {
                // Como se trada de una opcion
                // con argumento, verificamos que
                // el mismo haya sido provisto.
                //
                if (arg == 0) {
                    cerr << "Option requires argument: "
                    << "--"
                    << opt
                    << "\n";
                    exit(1);
                }
                op->parse(string(arg));
                return 1;
            } else {
                // Opcion sin argumento.
                //
                op->parse(string(""));
                return 0;
            }
        }
    }

    // Error: opcion no reconocida. Imprimimos un mensaje
    // de error, y finalizamos la ejecucion del programa.
    //
    cerr << "Unknown option: "
    << "--"
    << opt
    << "."
    << endl;
    exit(1);

    // Algunos compiladores se quejan con funciones que
    // logicamente no pueden terminar, y que no devuelven
    // un valor en esta ultima parte.
    //
    return -1;
}

int
cmdline::do_short_opt(const char *opt, const char *arg)
{
    option_t *op;

    // Recorremos la tabla de opciones, y buscamos la
    // entrada corta que se corresponda con la opcion de
```

```
    // linea de comandos. De no encontrarse, indicamos el
    // error.
    //
    for (op = option_table; op->short_name != 0; ++op) {
        if (string(opt) == string(op->short_name)) {
            // Marcamos esta opcion como usada en
            // forma explicita, para evitar tener
            // que inicializarla con el valor por
            // defecto.
            //
            op->flags |= OPT_SEEN;

            if (op->has_arg) {
                // Como se trata de una opcion
                // con argumento, verificamos que
                // el mismo haya sido provisto.
                //
                if (arg == 0) {
                    cerr << "Option requires argument: "
                    << "-"
                    << opt
                    << "\n";
                    exit(1);
                }
                op->parse(string(arg));
                return 1;
            } else {
                // Opcion sin argumento.
                //
                op->parse(string(""));
                return 0;
            }
        }
    }

    // Error: opcion no reconocida. Imprimimos un mensaje
    // de error, y finalizamos la ejecucion del programa.
    //
    cerr << "Unknown option: "
    << "-"
    << opt
    << "."
    << endl;
    exit(1);

    // Algunos compiladores se quejan con funciones que
    // logicamente no pueden terminar, y que no devuelven
    // un valor en esta ultima parte.
    //
    return -1;
}
```

precisionfija.h

```
#ifndef _PRECISIONFIJA_H_INCLUDED_
#define _PRECISIONFIJA_H_INCLUDED_

#include "bignum.h"
#include <regex>

struct precision_t {
  int value;
  bool isSet = false; // esta seteado por linea de argumento?
} ;

//manejo de los streams de entrada y salida, asigna los streams a los bignum
class precision_fija
{
    istream *iss_;
    ostream *oss_;
    precision_fija();

public:

    precision_fija(istream &,ostream &);
    ~precision_fija();
    void captura(precision_t *);
};

#endif
```

precisionfija.cpp

```
#include "precisionfija.h"

using namespace std;
precision_fija::precision_fija()
{
}
precision_fija::precision_fija(istream &iss, ostream &oss){
    iss_ = &iss;
    oss_ = &oss;
}
precision_fija::~precision_fija(){

}
void precision_fija::captura(precision_t *precision){

    string s; //se almacenar  la l na
    smatch m; // ver regex c++,
              //se almacenar  las "captura" realizadas por el regex
    static bool entry_error=false;

    //validacion regex---> https://regex101.com/
    //consultar: ^(\d+|-\d+|\+\d+)\s*(\+|\*|\-)\s*(\d+|\-\d+|\+\d+)$
    //\s --> matches any whitespace character (equivalent to [\r\n\t\f\v ])
    regex e ("^(\\d+|\\-\\d+|\\+\\d+)\\s*(\\+|\\*|\\-)\\s*(\\d+|\\-\\d+|\\+\\d+)$");

    // Recibo el flujo isstream y lo guarda en un string
    while (getline(*iss_, s)){
        if (s.empty() == true){
                *oss_<<"Finished program"<<endl;
                break;
        }
        if (std::regex_search (s,m,e)) {

            string a=m.str(1);
            string b=m.str(3);

            int precision_a = precision->isSet ? precision->value:a.length();
            int precision_b = precision->isSet ? precision->value:b.length();
            //cout<<precision_a<<" "<<precision_b<<endl;
            bignum aa(a, precision_a);
            bignum bb(b, precision_b);
            switch (m.str(2)[0])
            {
            case '+':
                *oss_ << aa + bb;
                break;
            case '-':
                *oss_ << aa - bb;
                break;
```

```
            case '*':
                *oss_ << aa * bb;
                break;
            default:
                // validado desde el regex, no es posible que llegue ac ;
                break;
            }
            s = m.suffix().str();
        }
        else{
            *oss_<<"Entry not processed"<<endl;
            entry_error=true;
        }
    }
    if (iss_->bad()) {
        cerr << "cannot read from input stream."
        << endl;
        exit(1);
    }
    if(entry_error) {
        exit(1); //el programa terminar  con valor no nulo ya que
                 //hubo al menos un error en el procesamiento de operaciones.
    }
}
```

tp0.cpp

```
#include <fstream>
#include <iomanip>
#include <iostream>
#include <sstream>
#include <string>
#include <cstdlib>
#include "bignum.h"
#include "cmdline.h"
#include "precisionfija.h"

static void opt_input(string const &);
static void opt_output(string const &);
static void opt_help(string const &);
static void opt_precision(string const &);

static option_t options[] = {
    {1, "p", "precision",NULL, opt_precision, OPT_DEFAULT},
    {1, "i", "input",  "-", opt_input,  OPT_DEFAULT},
    {1, "o", "output", "-", opt_output, OPT_DEFAULT},
    {0, "h", "help",  NULL, opt_help,   OPT_DEFAULT},
    {0, },
};

static istream *iss = 0;
static ostream *oss = 0;
static fstream ifs;
static fstream ofs;

// La precision si no es especificada el ejecturar el programa,
//se ajustara de acuerdo a la dimensi n del string que se asigna a bignum
static precision_t precision;

static void
opt_precision(string const &arg)
{
    //-p es mandatory
    // Validamos el arg de entrada
    // Iteramos sobre el string arg en busqueda de argumentos no numericos
    string::const_iterator it = arg.begin();
    while(it != arg.end() && std::isdigit(*it))
        ++it;
    if (!arg.empty() && it == arg.end()){
        precision.isSet=true;
        precision.value = std::stoi(arg); // transformamos a entero
        //cout<< "Es numerico"<<endl;
    }
    else{
        cerr << "La precisi n: "
        << arg
```

```
        << " No es un argumento n merico"
        << endl;
        exit(1);
    }
}

static void
opt_input(string const &arg)
{
    // Si el nombre del archivos es "-", usaremos la entrada
    // estandar. De lo contrario, abrimos un archivo en modo
    // de lectura.
    //
    if (arg == "-") {
        iss = &cin;
    } else {
        ifs.open(arg.c_str(), ios::in);
        iss = &ifs;
    }

    // Verificamos que el stream este OK.
    //
    if (!iss->good()) {
        cerr << "cannot open "
        << arg
        << "."
        << endl;
        exit(1);
    }
}

static void
opt_output(string const &arg)
{
    // Si el nombre del archivos es "-", usaremos la salida
    // estandar. De lo contrario, abrimos un archivo en modo
    // de escritura.
    //
    if (arg == "-") {
        oss = &cout;
    } else {
        ofs.open(arg.c_str(), ios::out);
        oss = &ofs;
    }

    // Verificamos que el stream este OK.
    //
    if (!oss->good()) {
        cerr << "cannot open "
        << arg
        << "."
```

```
        << endl;
        exit(1);
    }
}

static void
opt_help(string const &arg)
{
    cout << "tp0 [-p precision] [-i file] [-o file]"
    << endl;
    exit(0);
}


int
main(int argc, char * const argv[])
{
    cmdline cmdl(options);
    cmdl.parse(argc, argv);

    precision_fija precision_(*iss, *oss);
    precision_.captura(&precision);
    if (iss->bad()) {
        cerr << "cannot read from input stream."
        << endl;
        exit(1);
    }
    return 0;
}
```