Programmazione (CL.B)

Elaborato 10

Andrea Piroddi

Dipartimento di Informatica, Scienza e Ingegneria

Elaborato 10

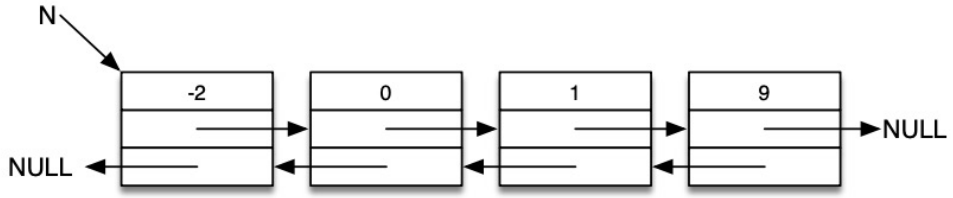
Sviluppare una funzione di libreria per la moltiplicazione di interi (con segno) di lunghezza arbitraria (***bigint***), rappresentati tramite liste concatenate.

```
bigint *mul(bigint *N1, bigint *N2);
```

- La funzione ***mul*** ritorna **NULL** se almeno uno dei due argomenti punta a **NULL**.
- La funzione ***mul*** deve essere sviluppata per la rappresentazione di un bigint tramite liste doppiamente concatenate (in ***dl_bigint.c***) e tramite liste doppiamente concatenate circolari (in ***cl bigint.c***).
- Per lo sviluppo della funzione ***mul*** `e possibile utilizzare (senza nemmeno dover apportare modifiche) qualsiasi funzione possa risultare utile tra quelle fornite come soluzioni per gli appelli dell'AA 2019-2020

Elaborato 10

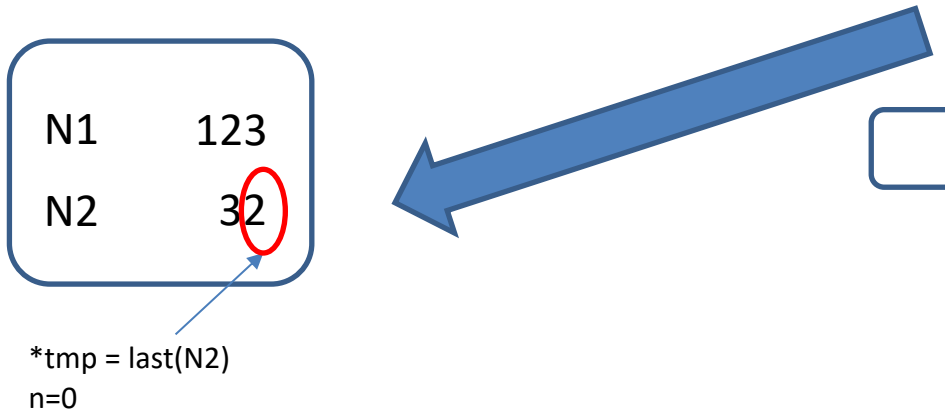
Rappresentazione di un ***bigint*** tramite liste doppiamente concatenate (***`dl_bigint.c`***)

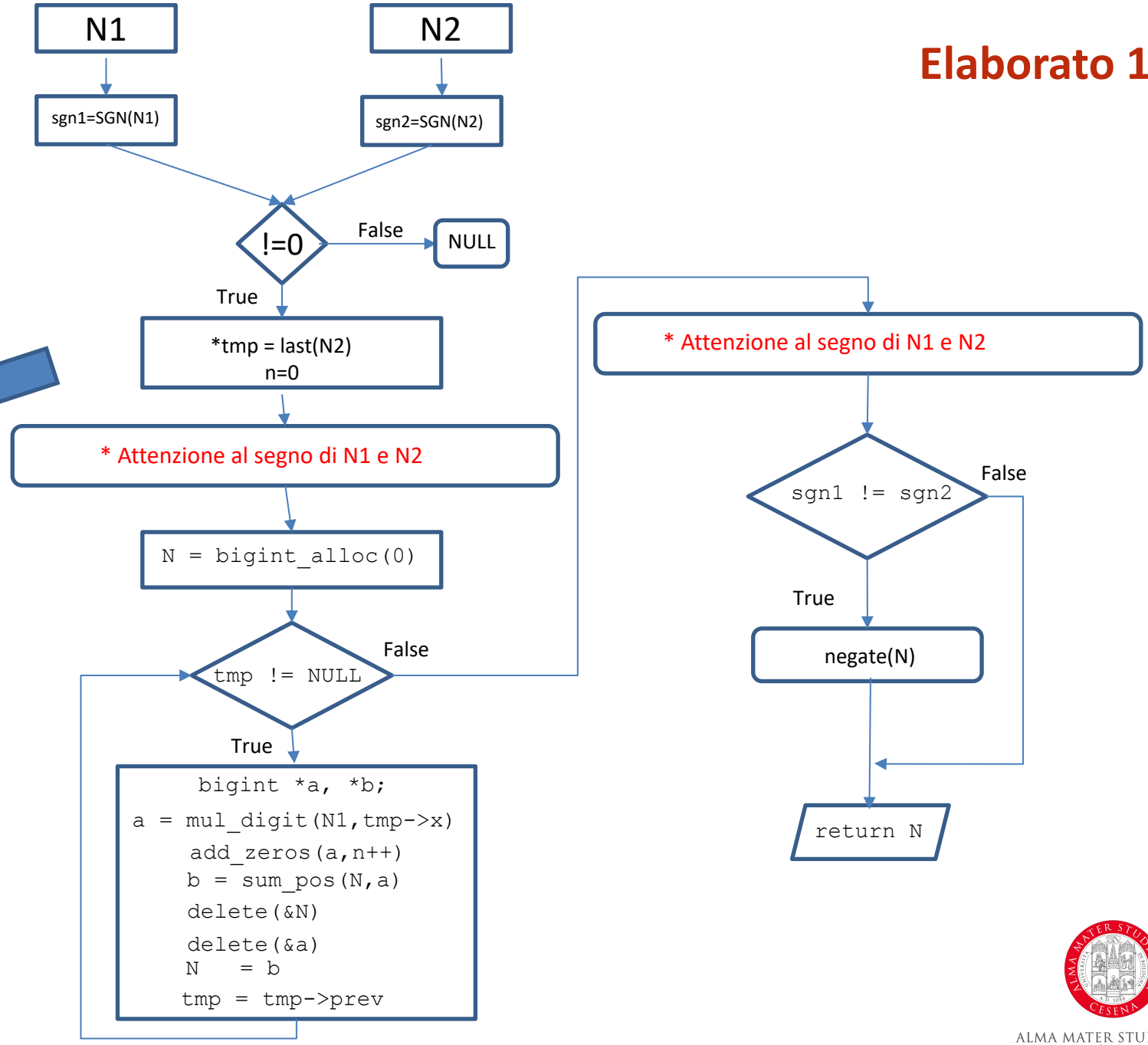



```c
typedef signed char digit;

typedef struct bigint {
  digit x;
  struct bigint *next;
  struct bigint *prev;
} bigint;
```

Elaborato 10

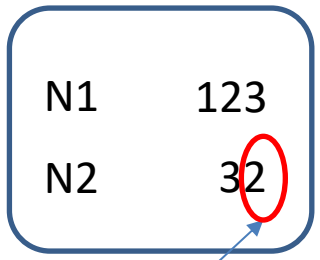
```
bigint *mul(bigint *N1, bigint *N2) {
        int sgn1 = SGN(N1), sgn2 = SGN(N2);
        bigint *N = NULL;
```





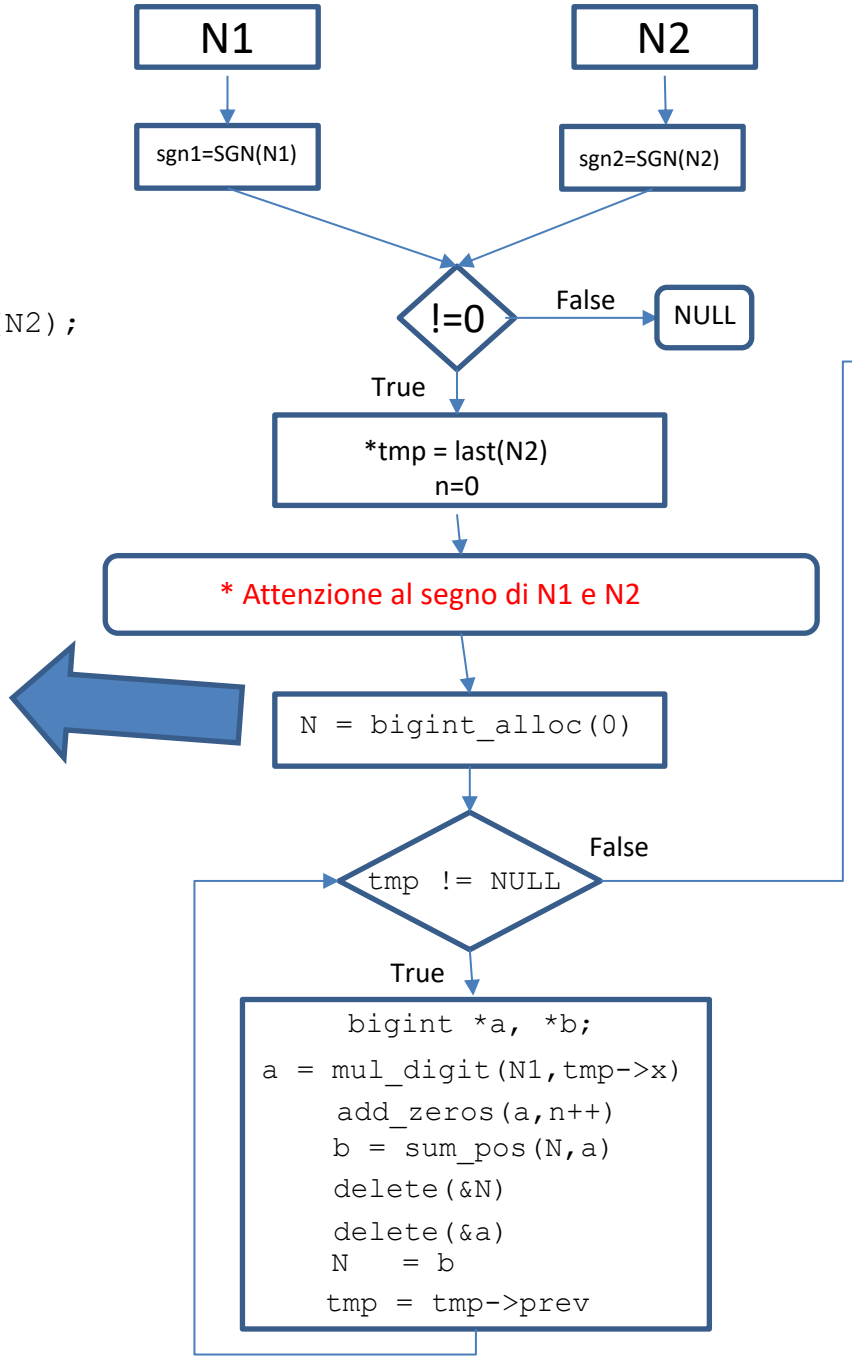
Elaborato 10

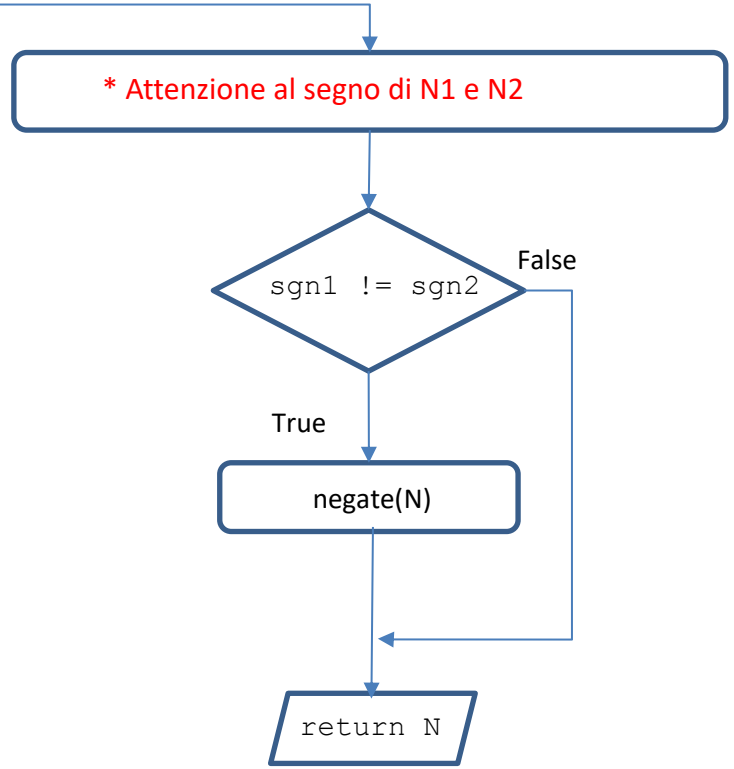
```
bigint *mul(bigint *N1, bigint *N2) {
        int sgn1 = SGN(N1), sgn2 = SGN(N2);
        bigint *N = NULL;
```



*tmp = last(N2)
n=0

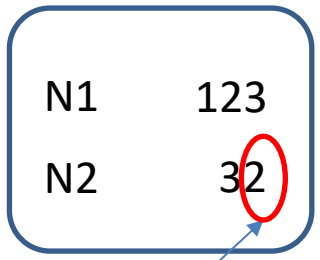
N = 0





Elaborato 10

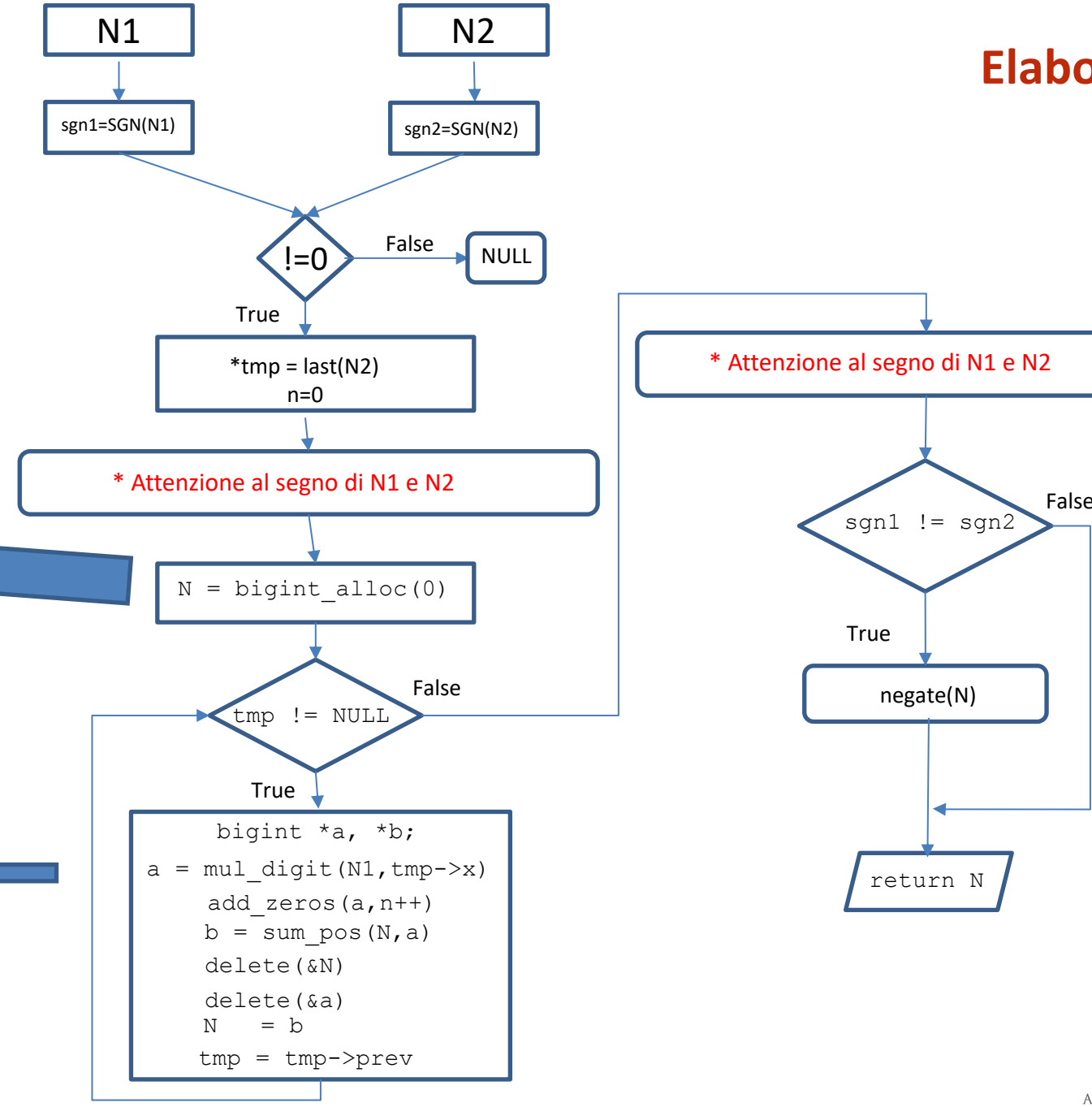
```
bigint *mul(bigint *N1, bigint *N2) {
        int sgn1 = SGN(N1), sgn2 = SGN(N2);
        bigint *N = NULL;
```



*tmp = last(N2)
n=0

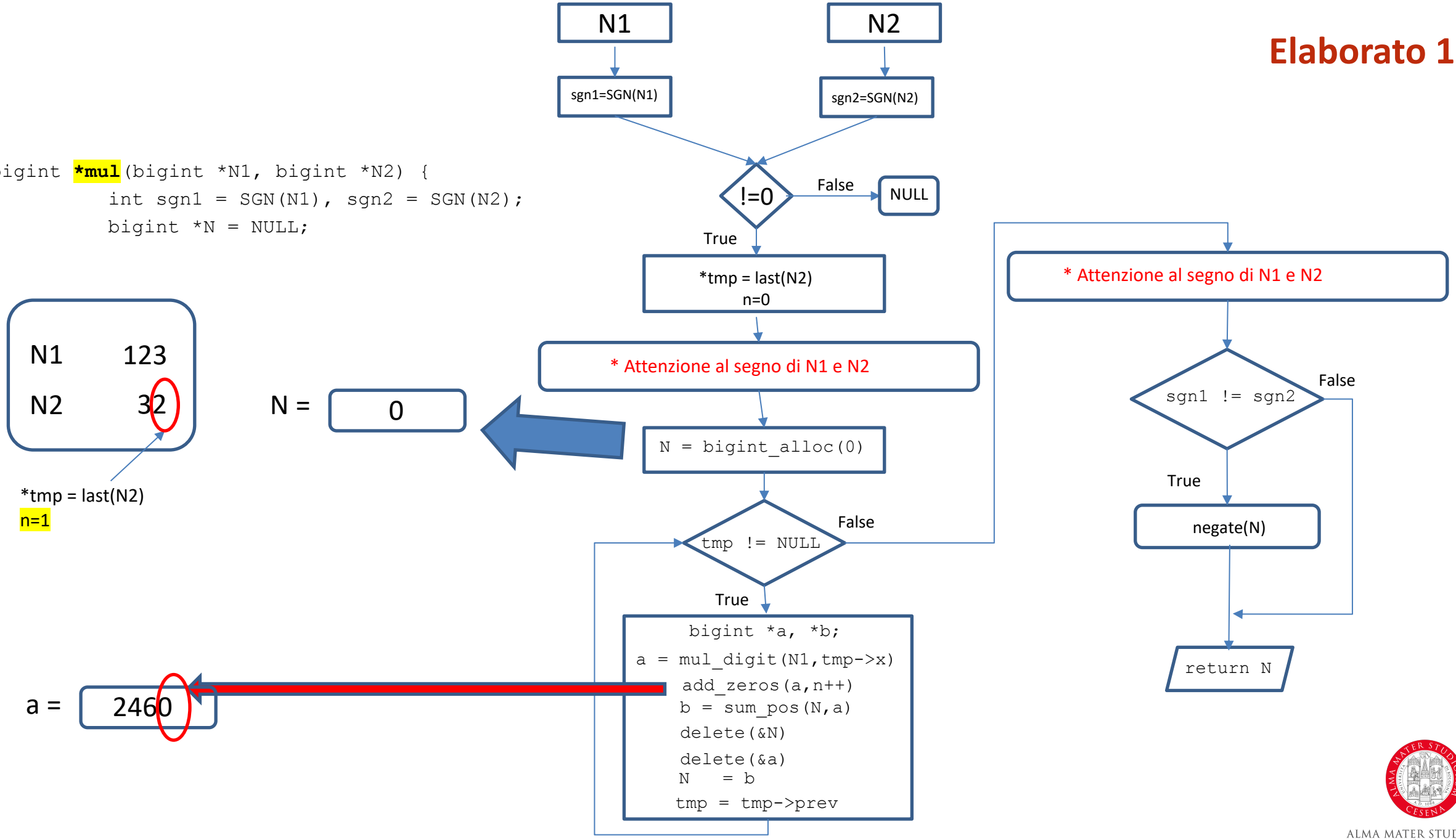
N = 0

a = 246



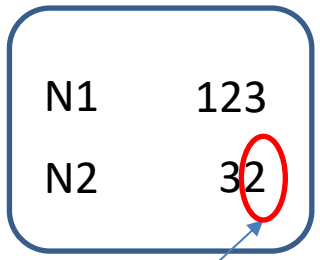
Elaborato 10

```
bigint *mul(bigint *N1, bigint *N2) {
        int sgn1 = SGN(N1), sgn2 = SGN(N2);
        bigint *N = NULL;
```



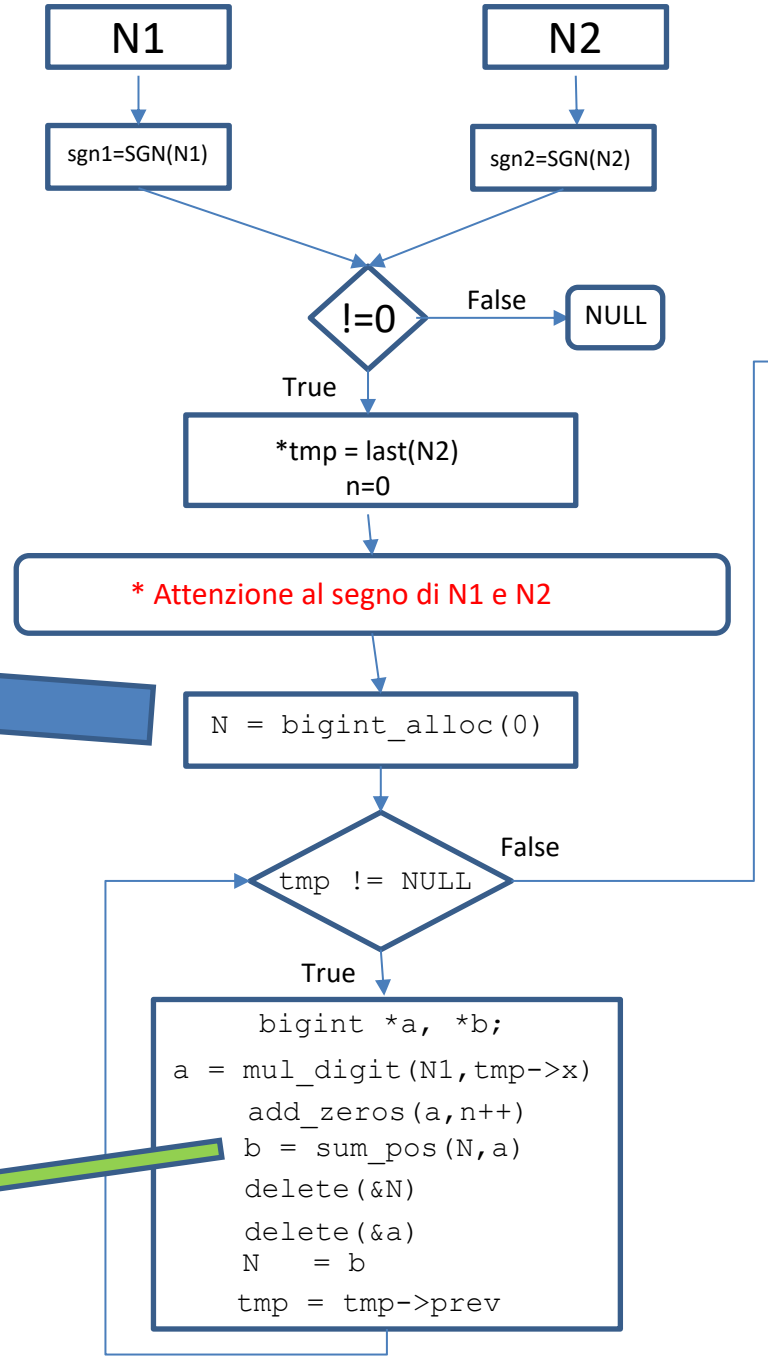
Elaborato 10

```
bigint *mul(bigint *N1, bigint *N2) {
        int sgn1 = SGN(N1), sgn2 = SGN(N2);
        bigint *N = NULL;
```



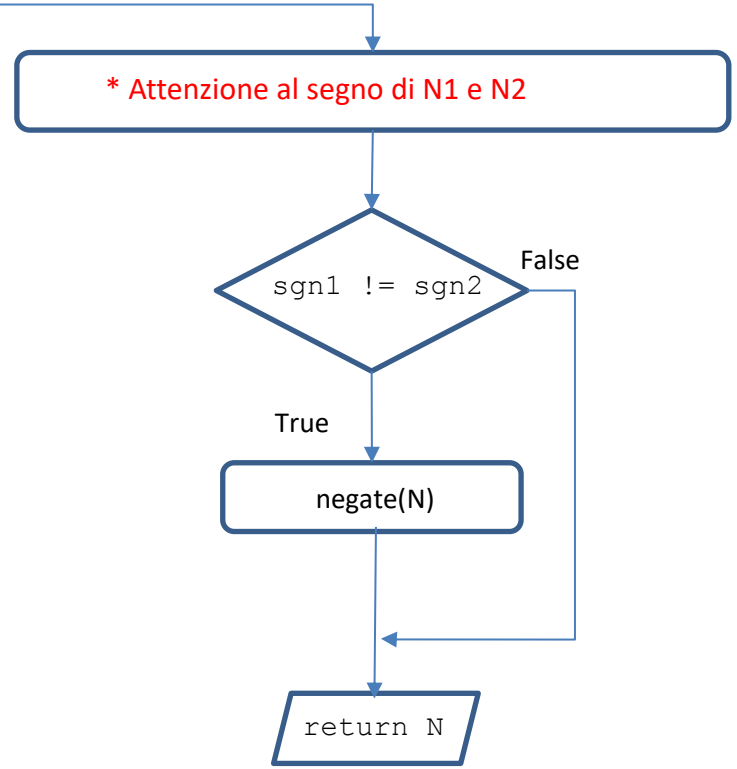
*tmp = last(N2)
n=1

N = 0



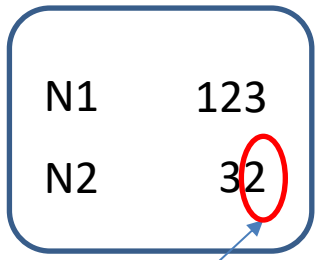
a = 2460

b = 246

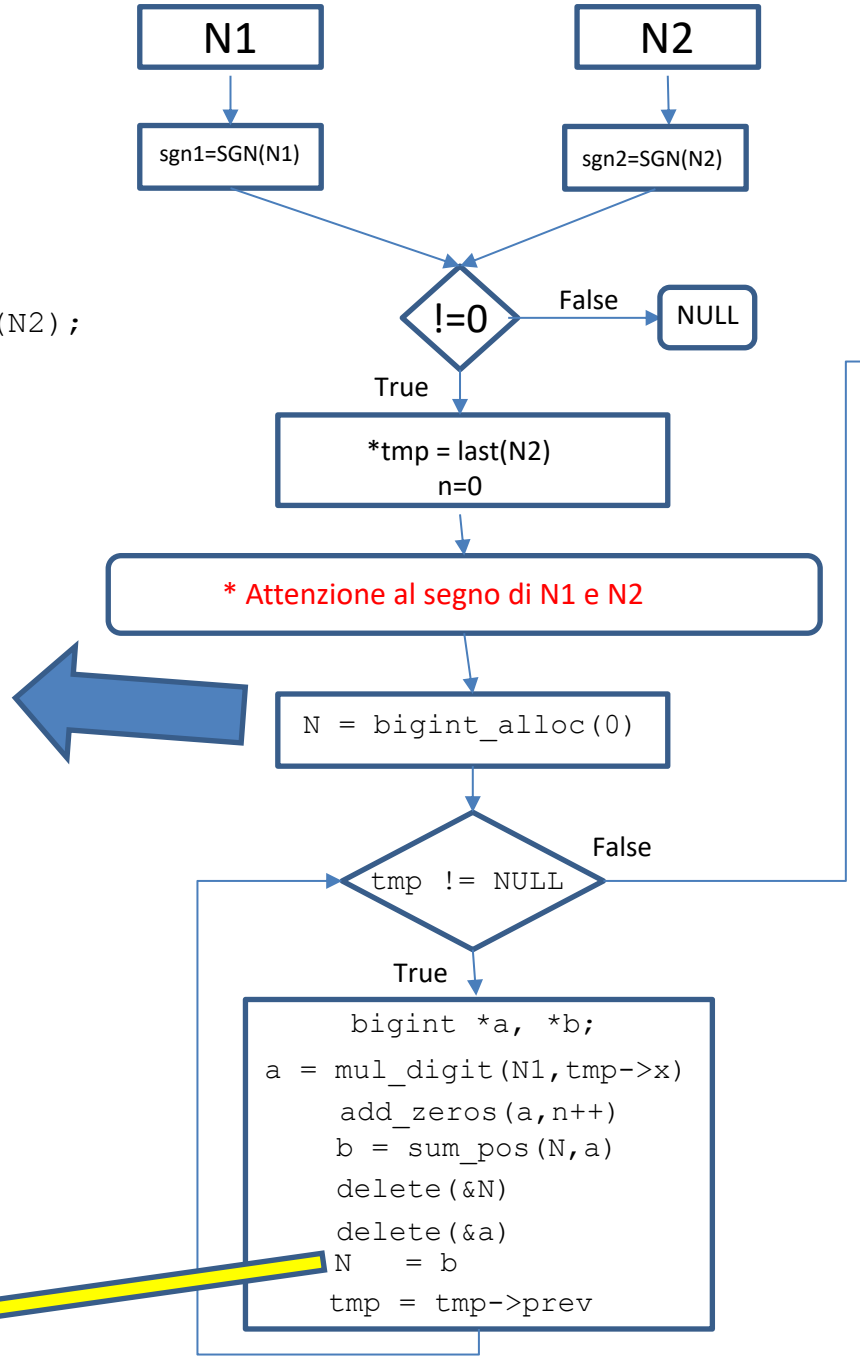


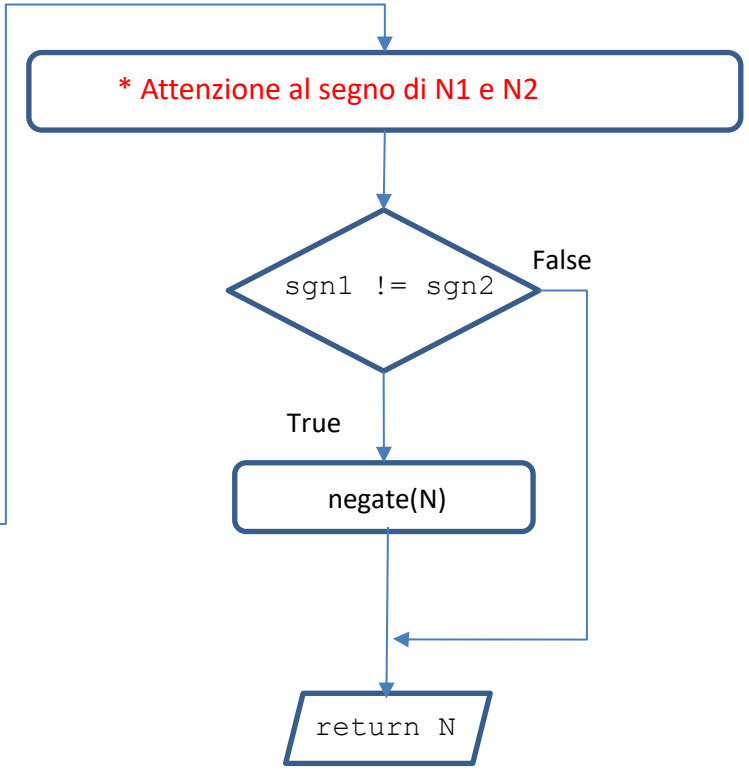
Elaborato 10

```
bigint *mul(bigint *N1, bigint *N2) {
        int sgn1 = SGN(N1), sgn2 = SGN(N2);
        bigint *N = NULL;
```



*tmp = last(N2)
n=1





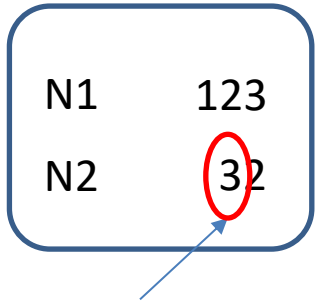
a =

b = 246

N = 246

Elaborato 10

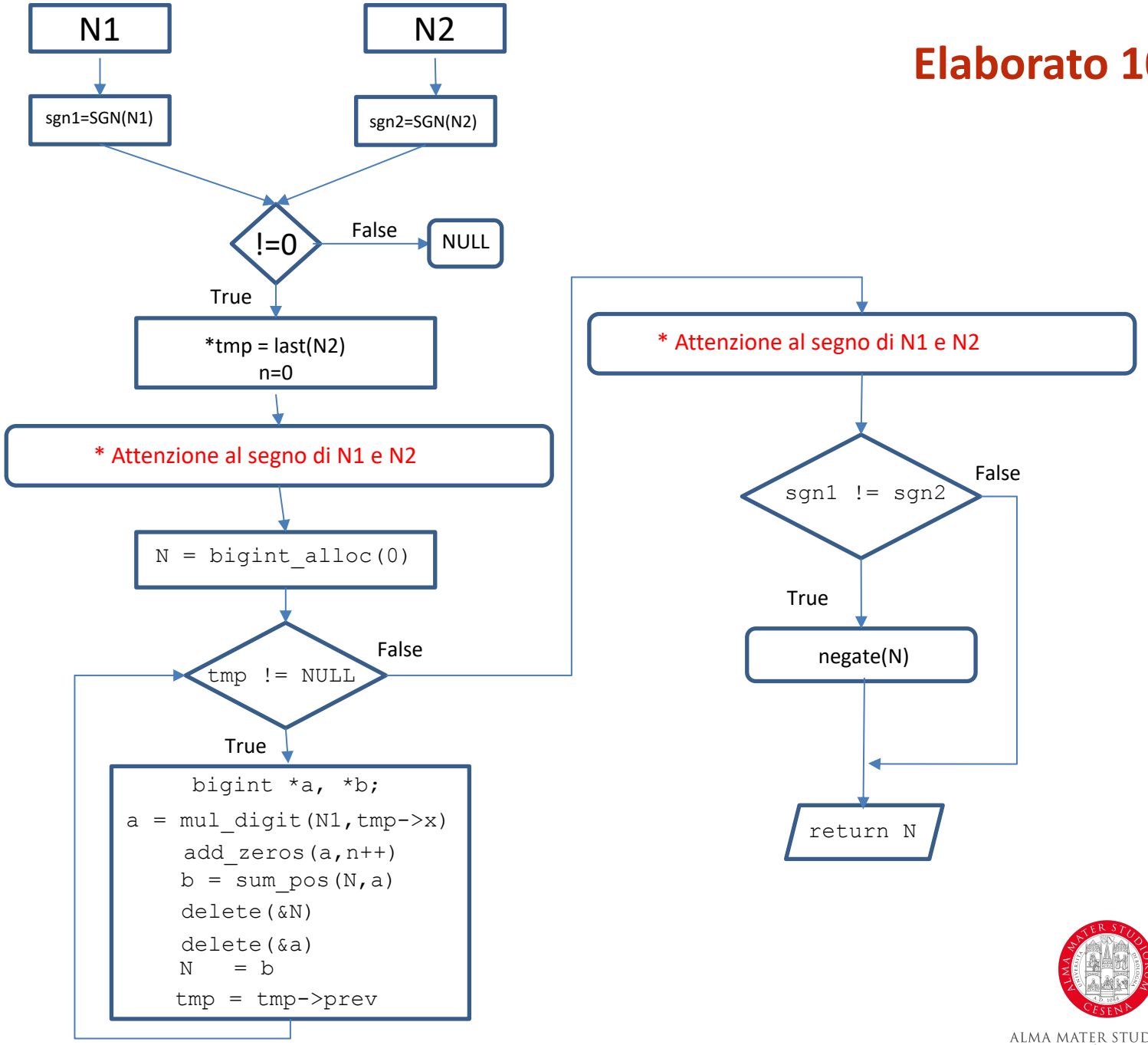
```
bigint *mul(bigint *N1, bigint *N2) {
        int sgn1 = SGN(N1), sgn2 = SGN(N2);
        bigint *N = NULL;
```



a =

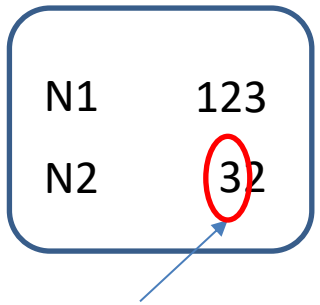
b = 246

N = 246

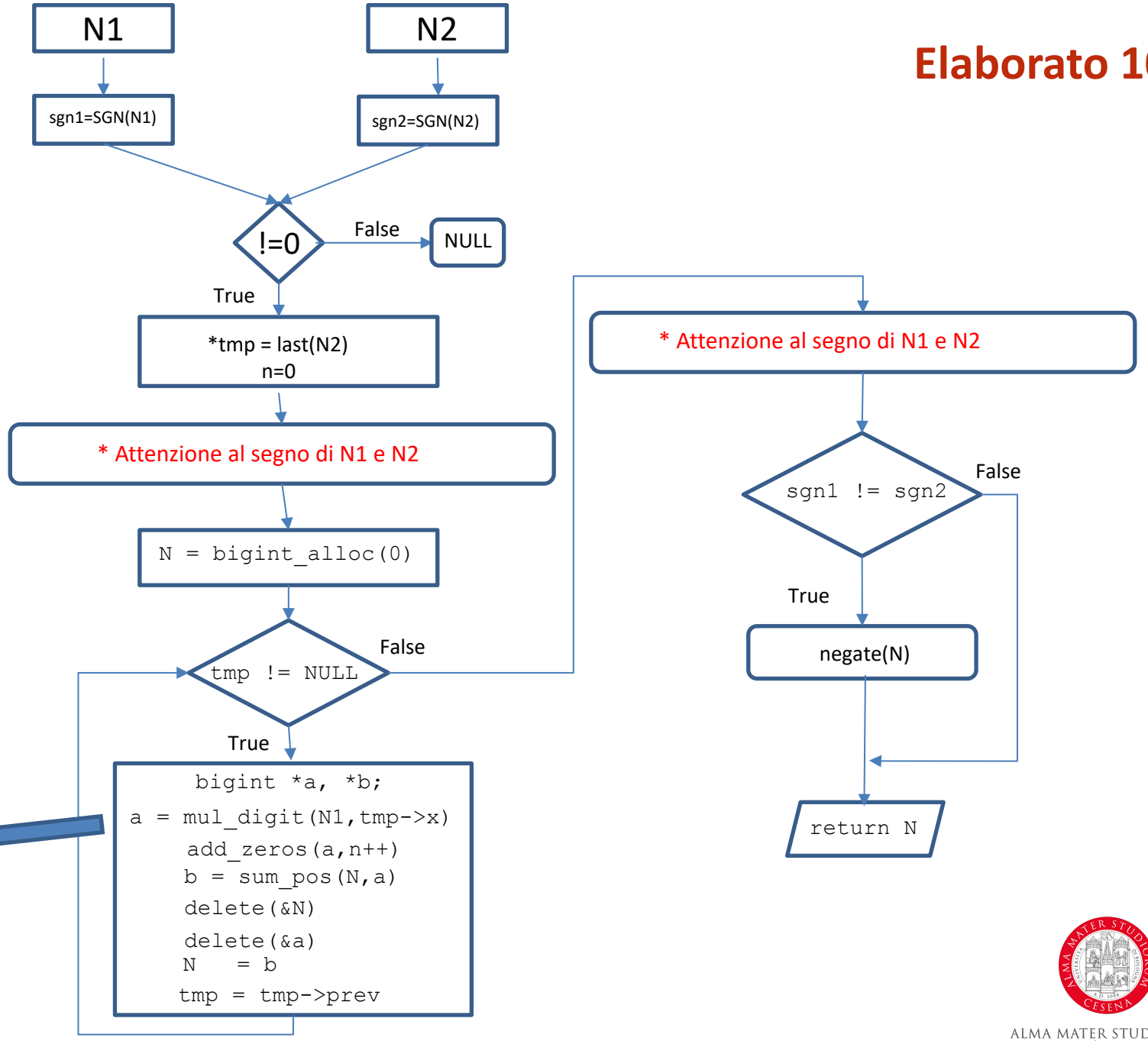


Elaborato 10

```
bigint *mul(bigint *N1, bigint *N2) {
        int sgn1 = SGN(N1), sgn2 = SGN(N2);
        bigint *N = NULL;
```

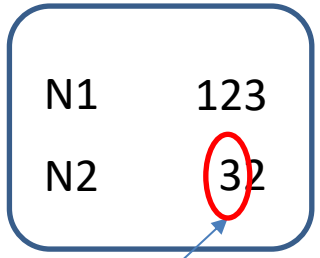


a = 369



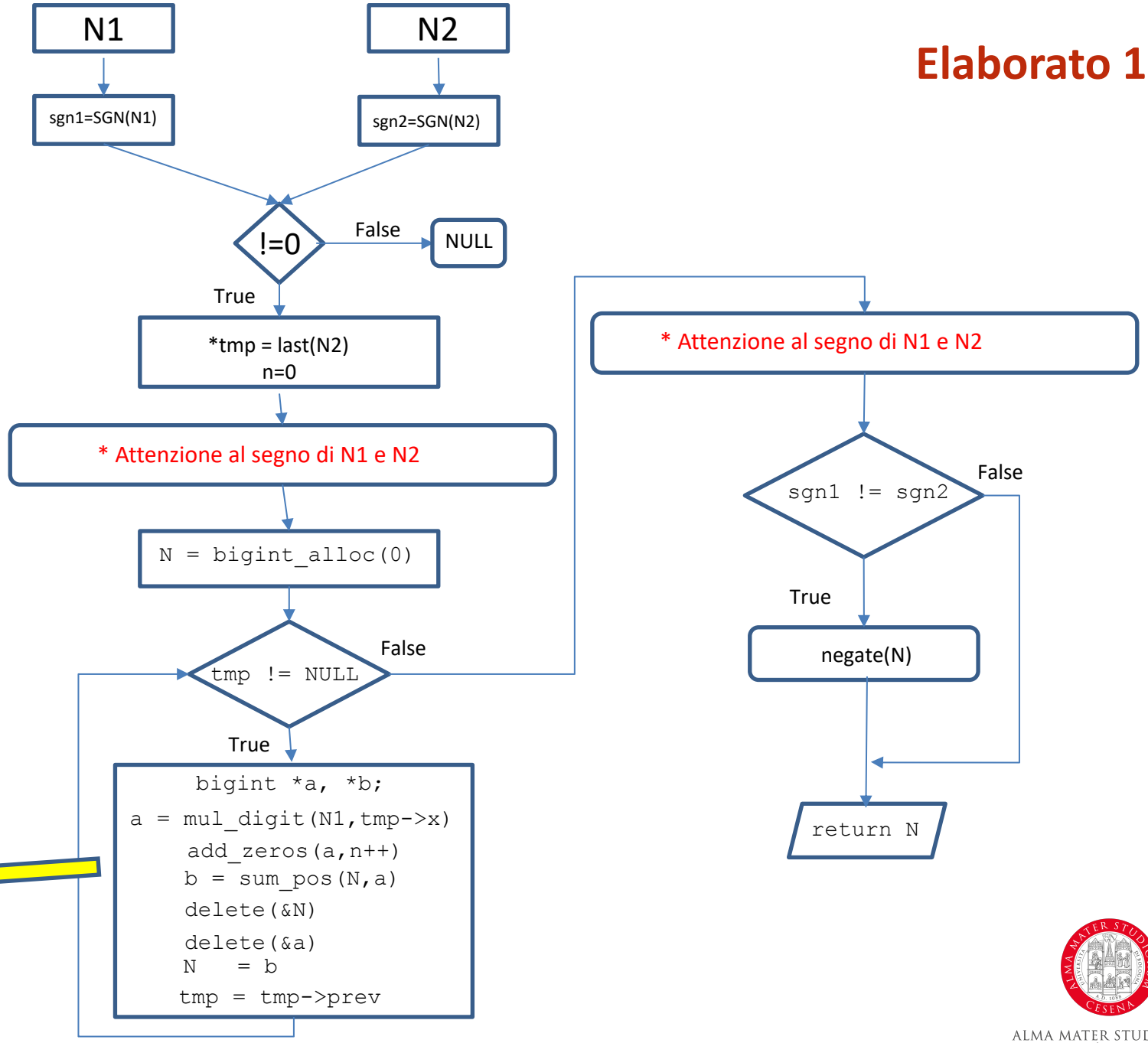
Elaborato 10

```
bigint *mul(bigint *N1, bigint *N2) {
        int sgn1 = SGN(N1), sgn2 = SGN(N2);
        bigint *N = NULL;
```



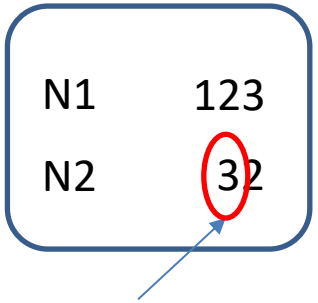
N = 246

a = 36900



Elaborato 10

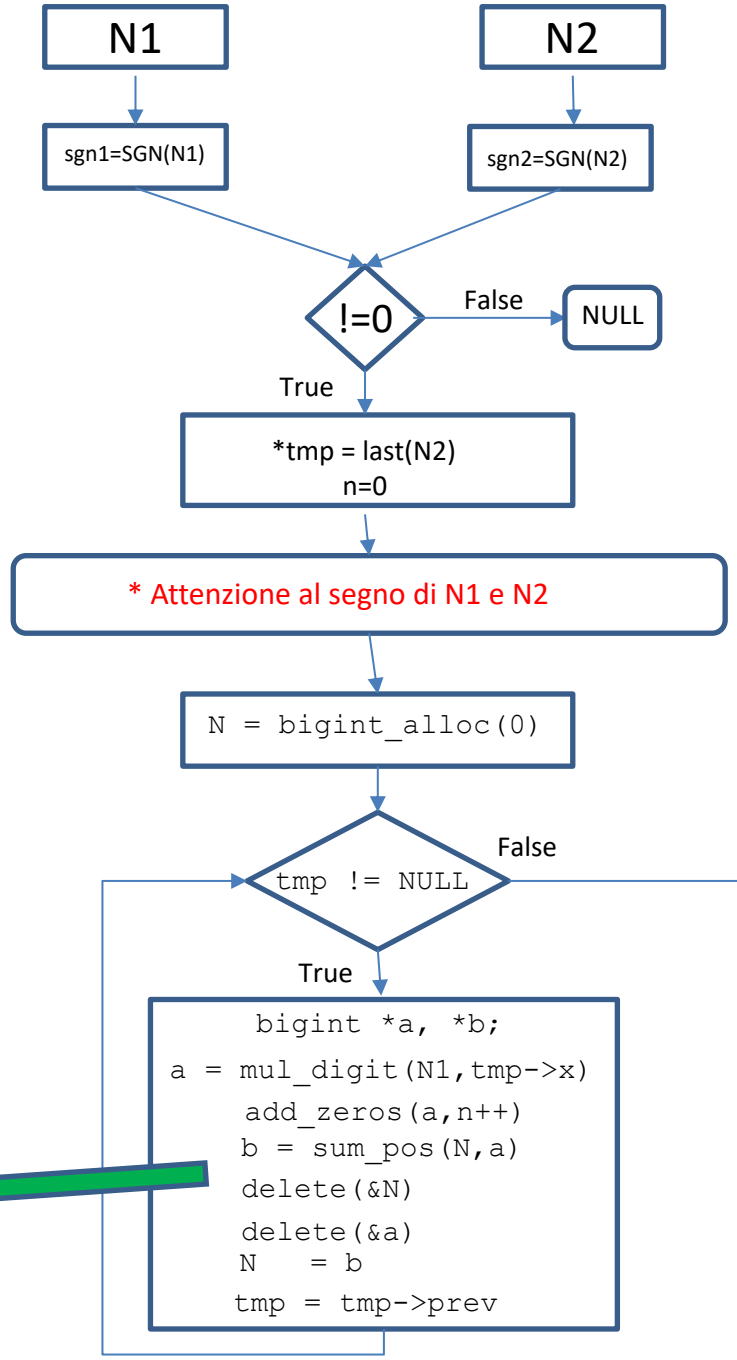
```
bigint *mul(bigint *N1, bigint *N2) {
        int sgn1 = SGN(N1), sgn2 = SGN(N2);
        bigint *N = NULL;
```

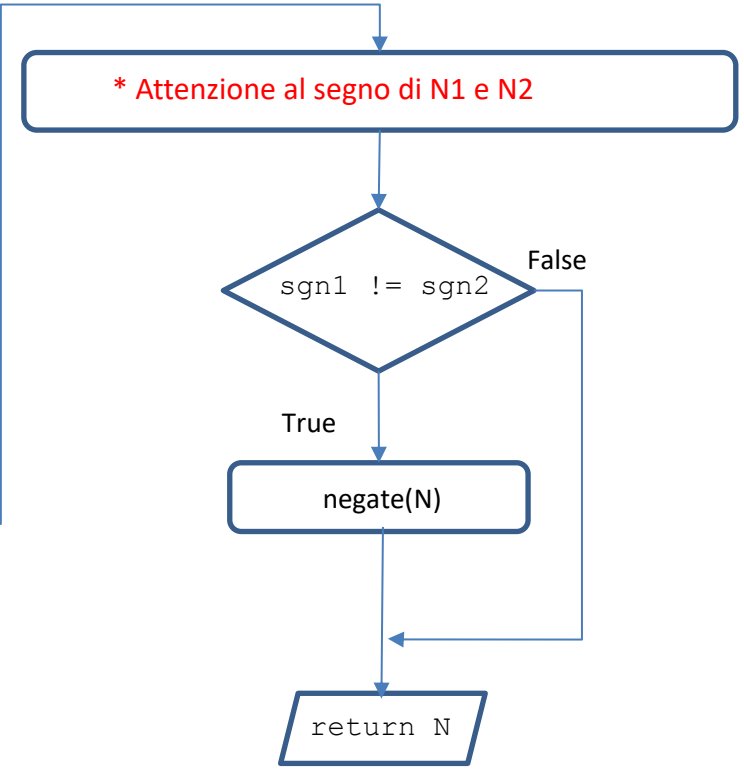


N = 246

a = 36900

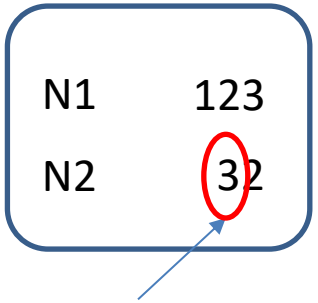
b = 246+3690





Elaborato 10

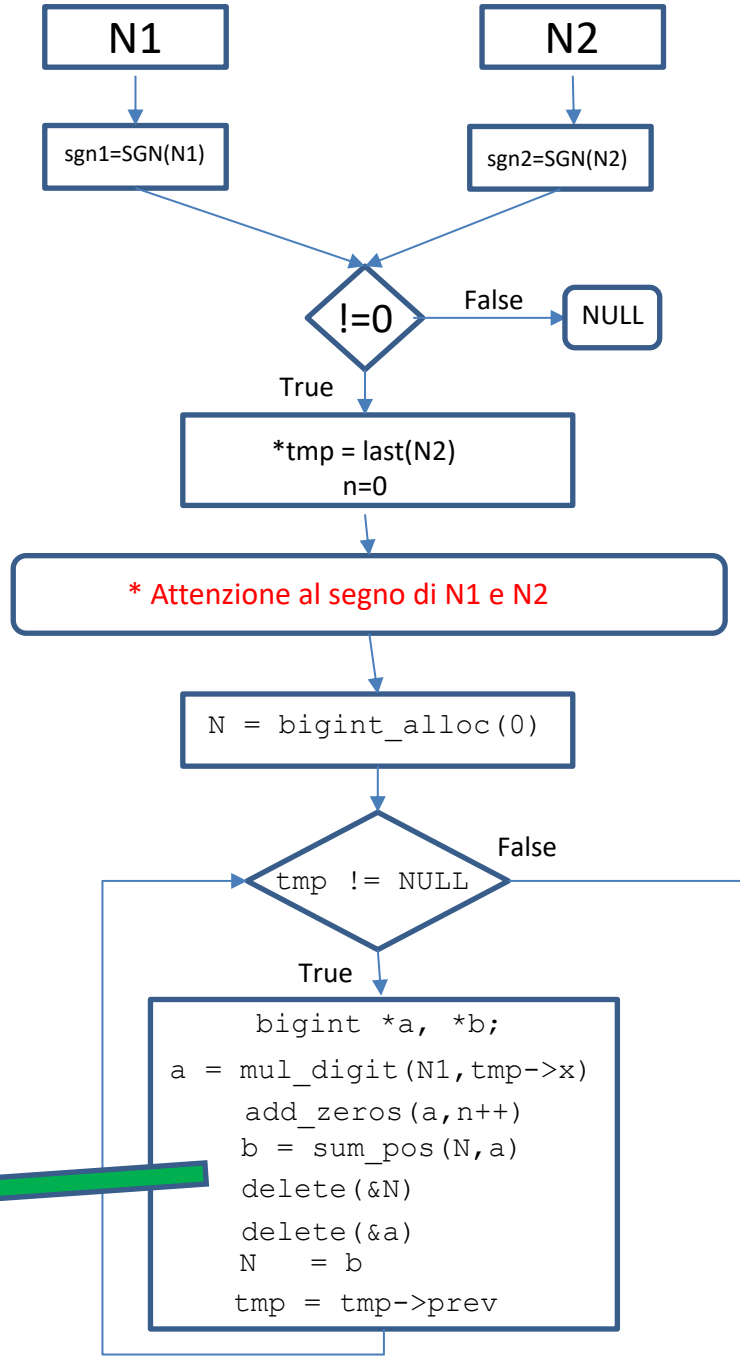
```
bigint *mul(bigint *N1, bigint *N2) {
        int sgn1 = SGN(N1), sgn2 = SGN(N2);
        bigint *N = NULL;
```

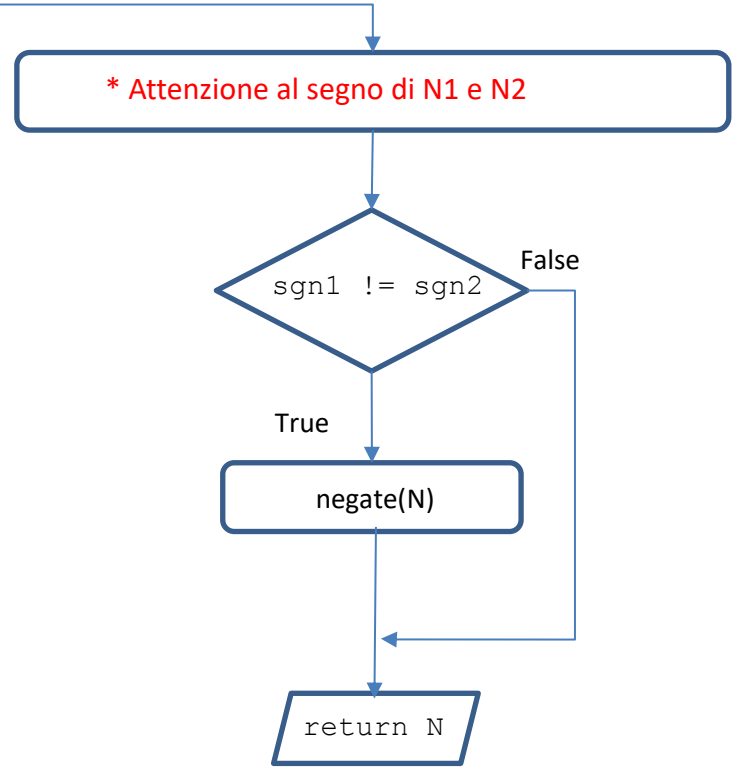


N = 2460

a = 36900

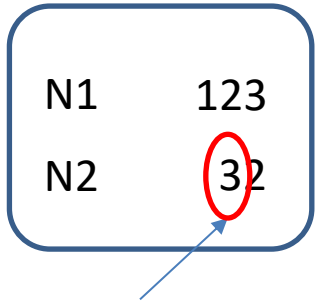
b = 3936





Elaborato 10

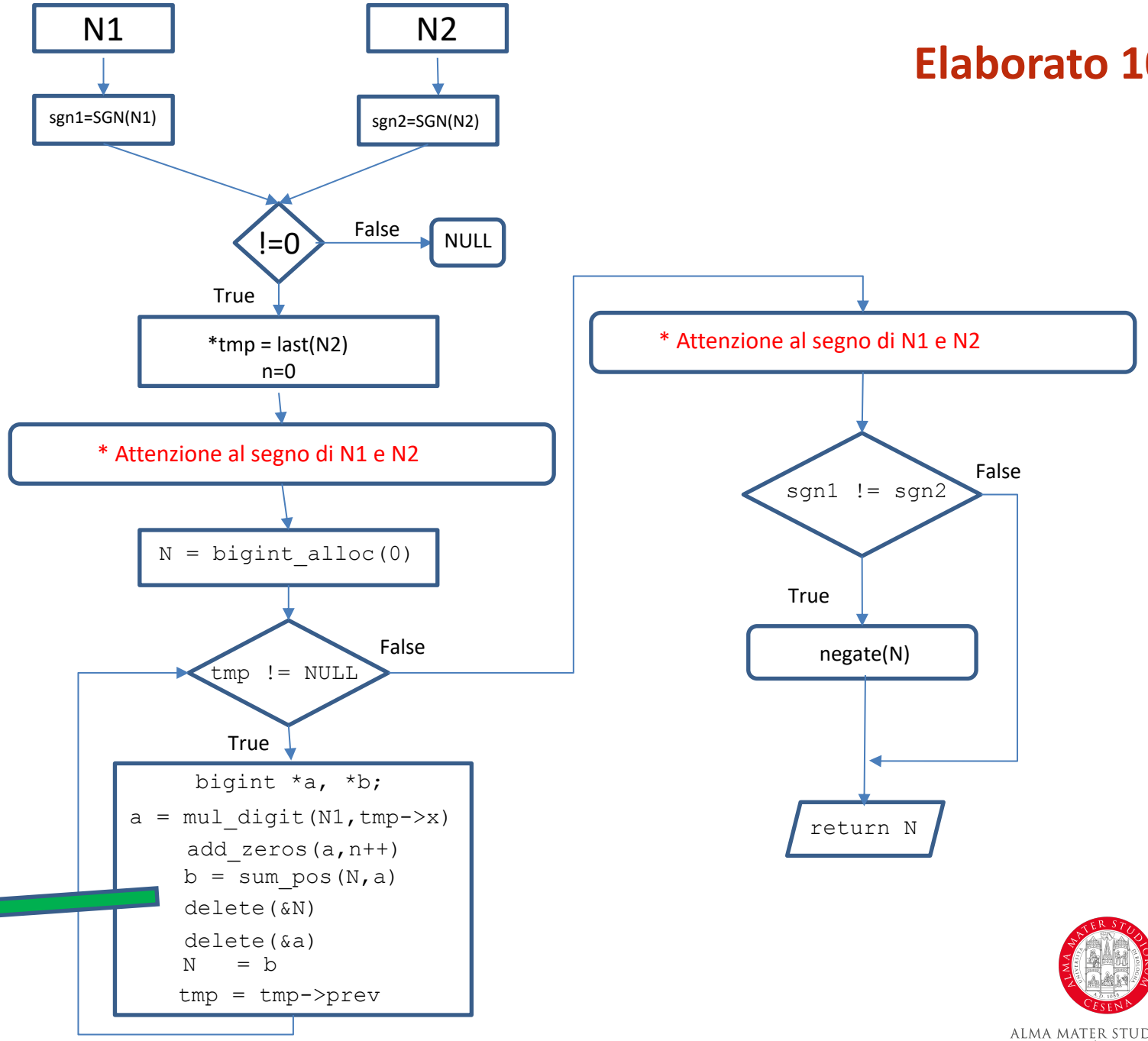
```
bigint *mul(bigint *N1, bigint *N2) {
        int sgn1 = SGN(N1), sgn2 = SGN(N2);
        bigint *N = NULL;
```



N = 2460

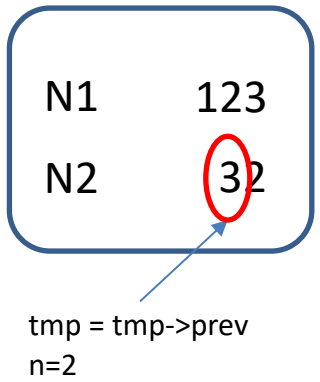
a = 36900

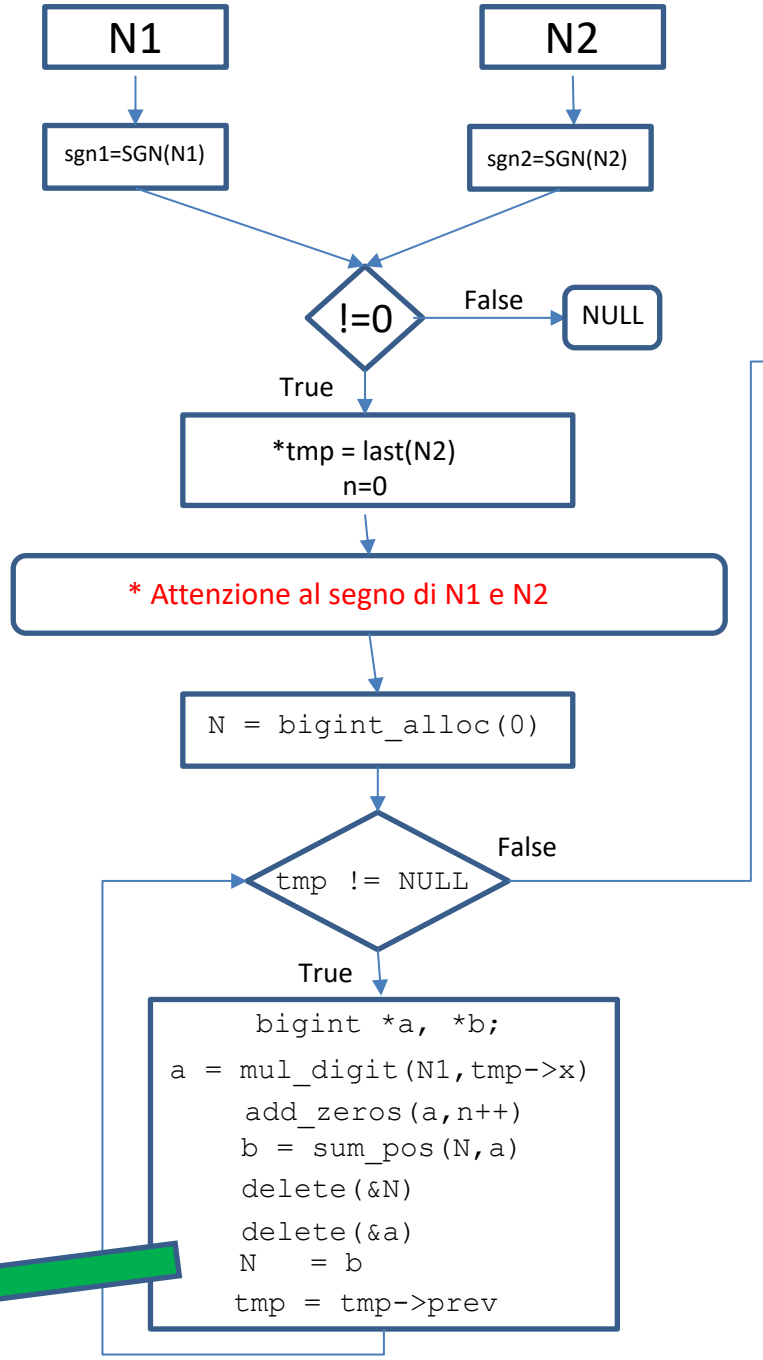
b = 3936

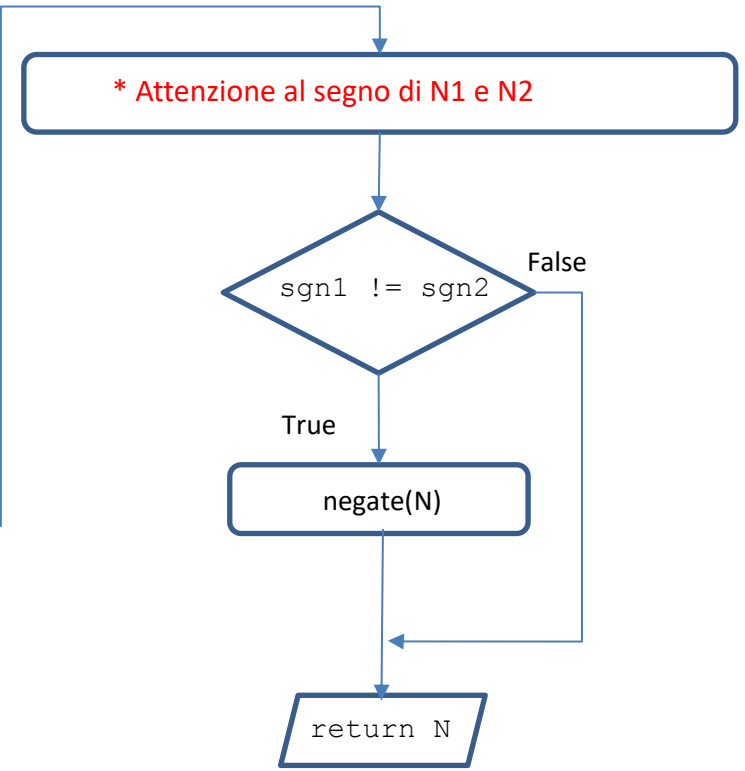


Elaborato 10

```
bigint *mul(bigint *N1, bigint *N2) {
        int sgn1 = SGN(N1), sgn2 = SGN(N2);
        bigint *N = NULL;
```







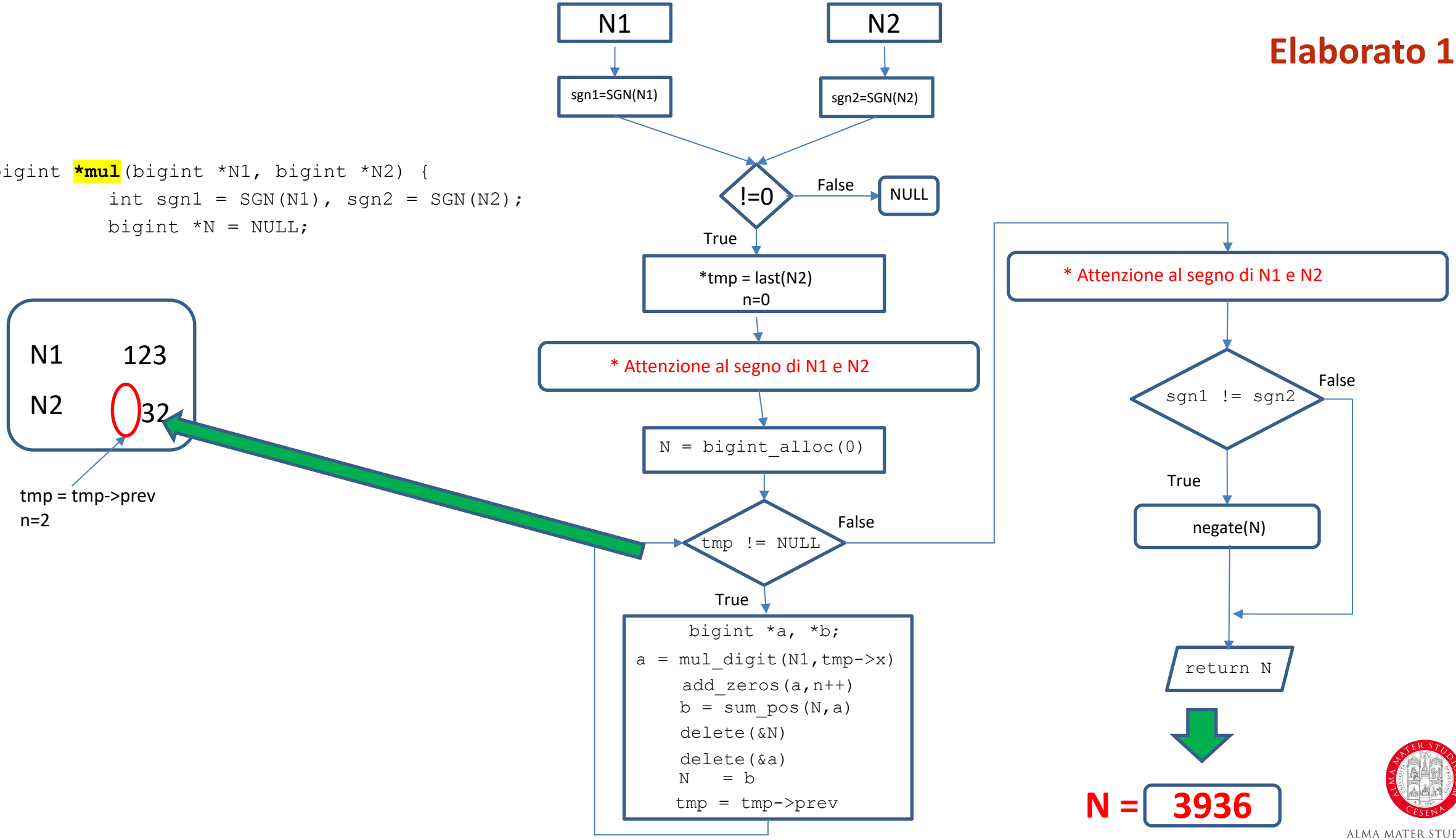
a = 36900

b = 3936

N = 3936

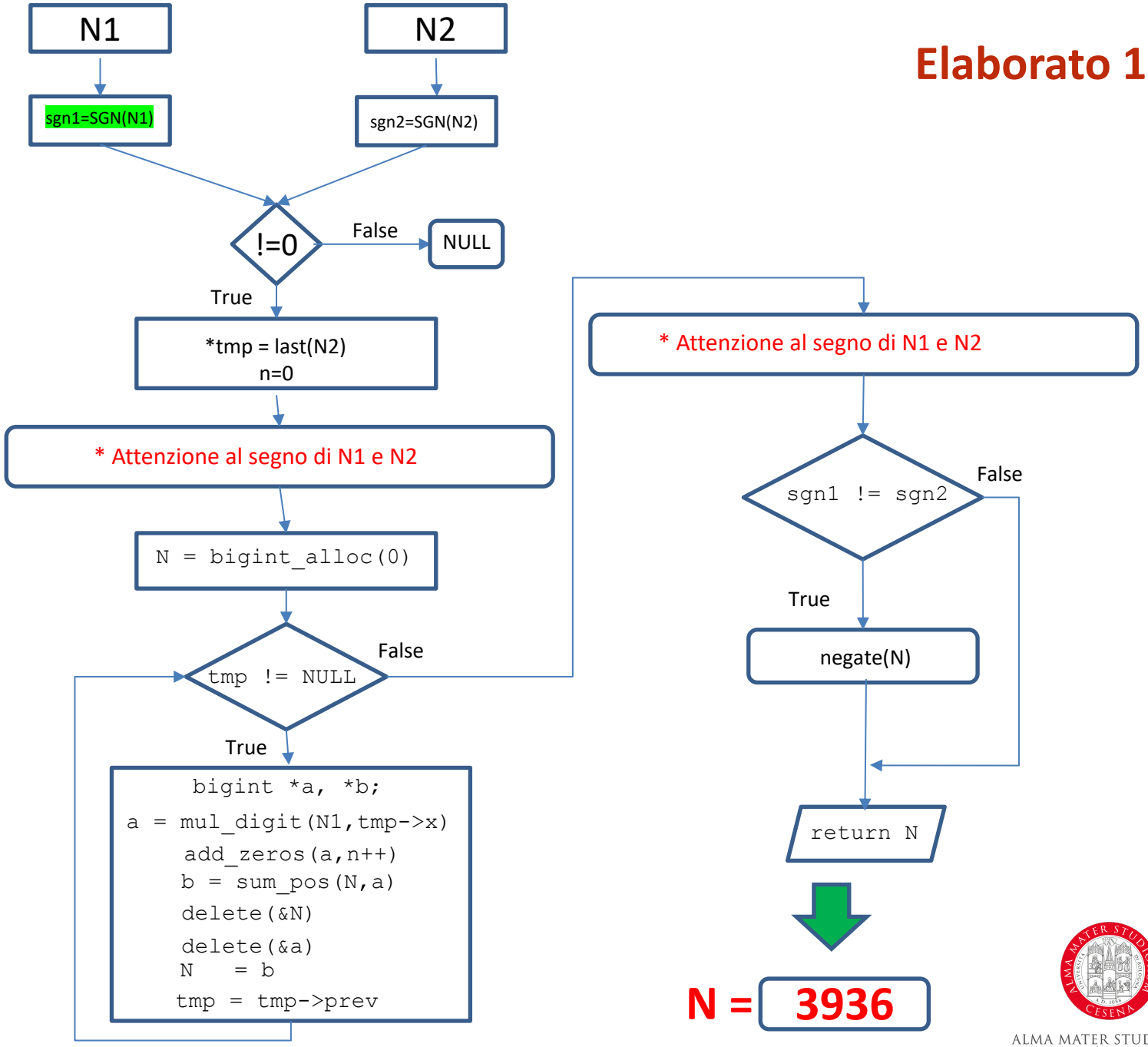
Elaborato 10

```
bigint *mul(bigint *N1, bigint *N2) {
        int sgn1 = SGN(N1), sgn2 = SGN(N2);
        bigint *N = NULL;
```



Analizziamo le funzioni

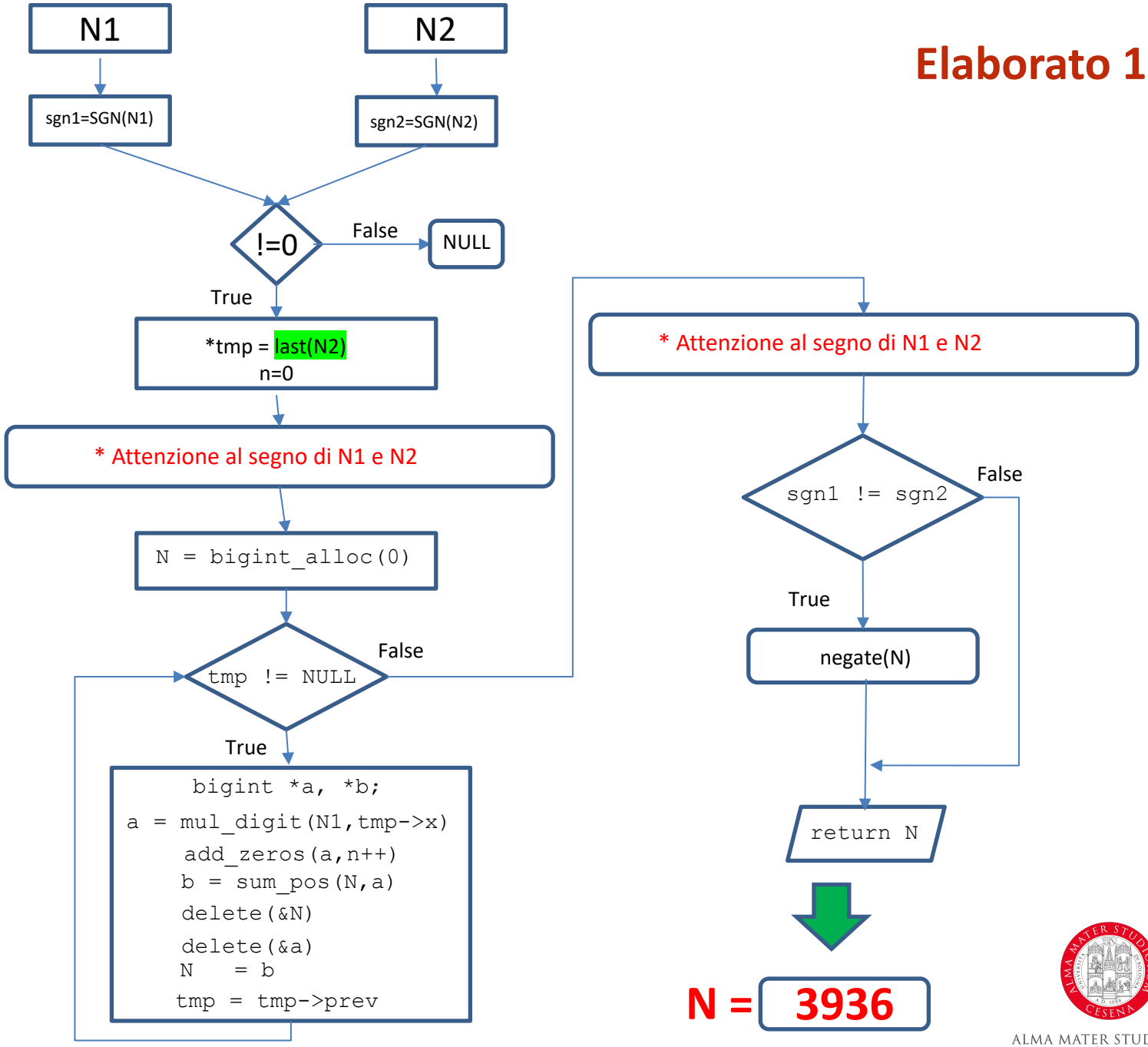
Elaborato 10



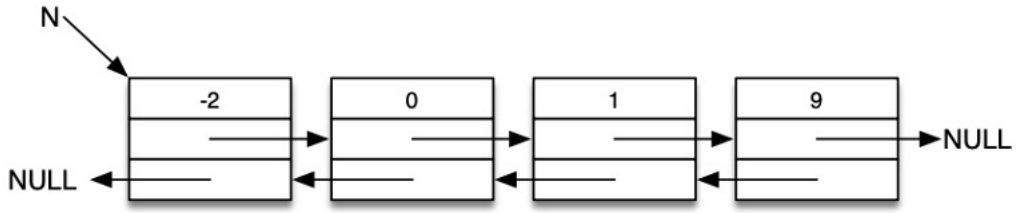
Analizziamo le funzioni - SGN(N)

```
#define SGN(N) ((N) == NULL ? 0 : ((N)->x < 0 ? -1 : 1))
```

Elaborato 10

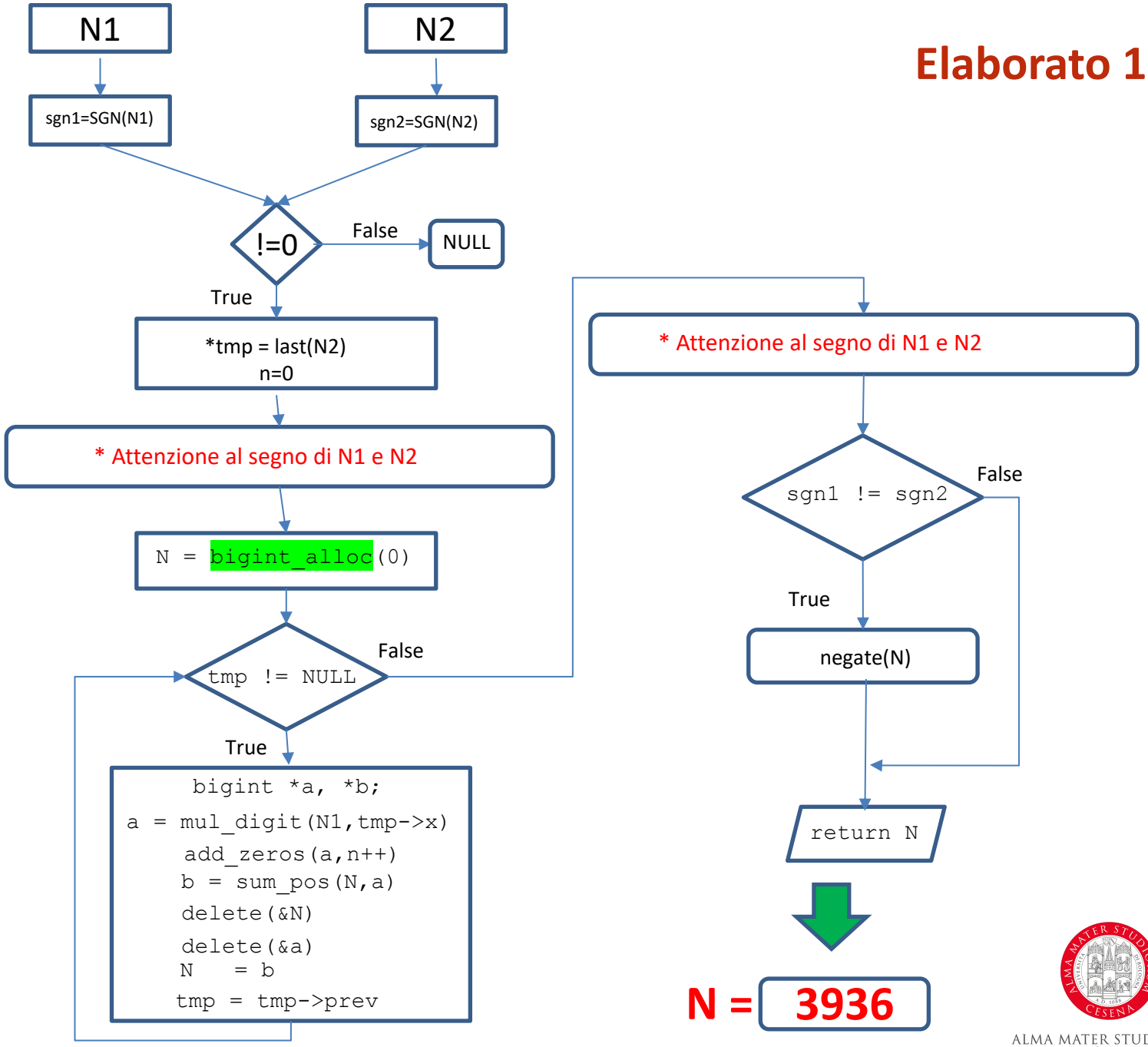


Analizziamo le funzioni – last(N)

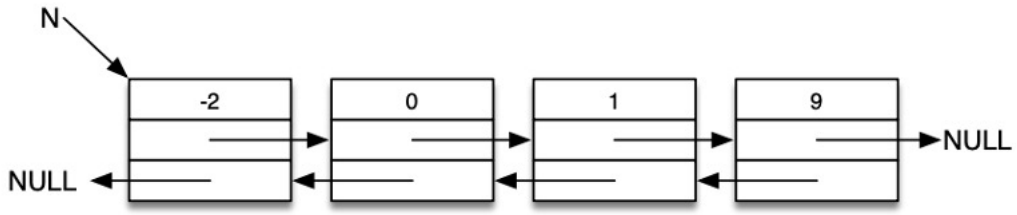



```
static bigint *last(bigint *N) {
        if(N != NULL) {
                // fintanto che il campo next di N è diverso da NULL
                // spostiamo N al successivo
                }
        return N;
}
```

Elaborato 10

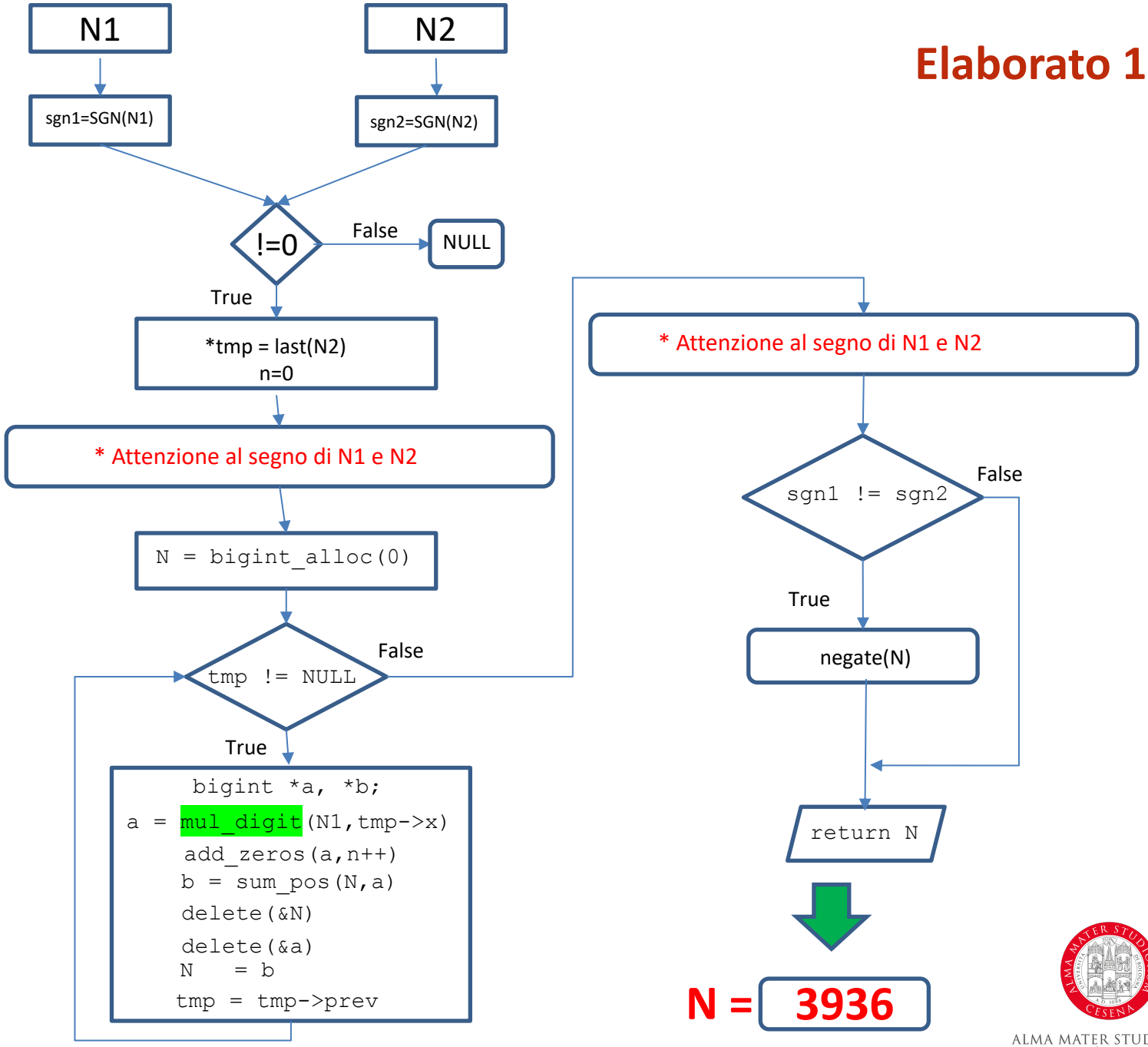


Analizziamo le funzioni – bigint_alloc




```
static bigint *bigint_alloc(digit x) {
  bigint *tmp = (bigint *)malloc(sizeof(bigint));

        if(tmp != NULL) {
                tmp->x    = x;
                tmp->next = NULL;
                tmp->prev = NULL;
        }
        return tmp;
}
```

Elaborato 10



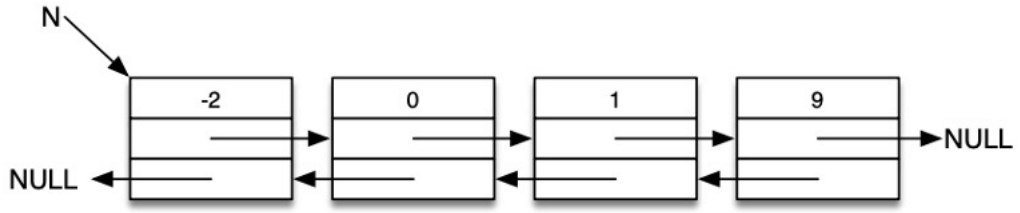
Analizziamo le funzioni – mul_digit

```
static bigint *mul_digit(bigint *N, digit x) {
        //se N è NULL o x è esterno all'intervallo (-9,9)
{       //restituisci NULL

        } //altrimenti se x è uguale a 0 {
          // chiama bigint_alloc(0);
        } //altrimenti {
                //definisci i segni di N e x
                // dichiara bigint *X;

                // tieni conto dei segni di N ed x

                // assegna ad X il valore della funzione mul_digit_pos(N,x);

                //definisci i segni di N ed X

                return X;
        }
}
```

Analizziamo le funzioni – mul_digit_pos

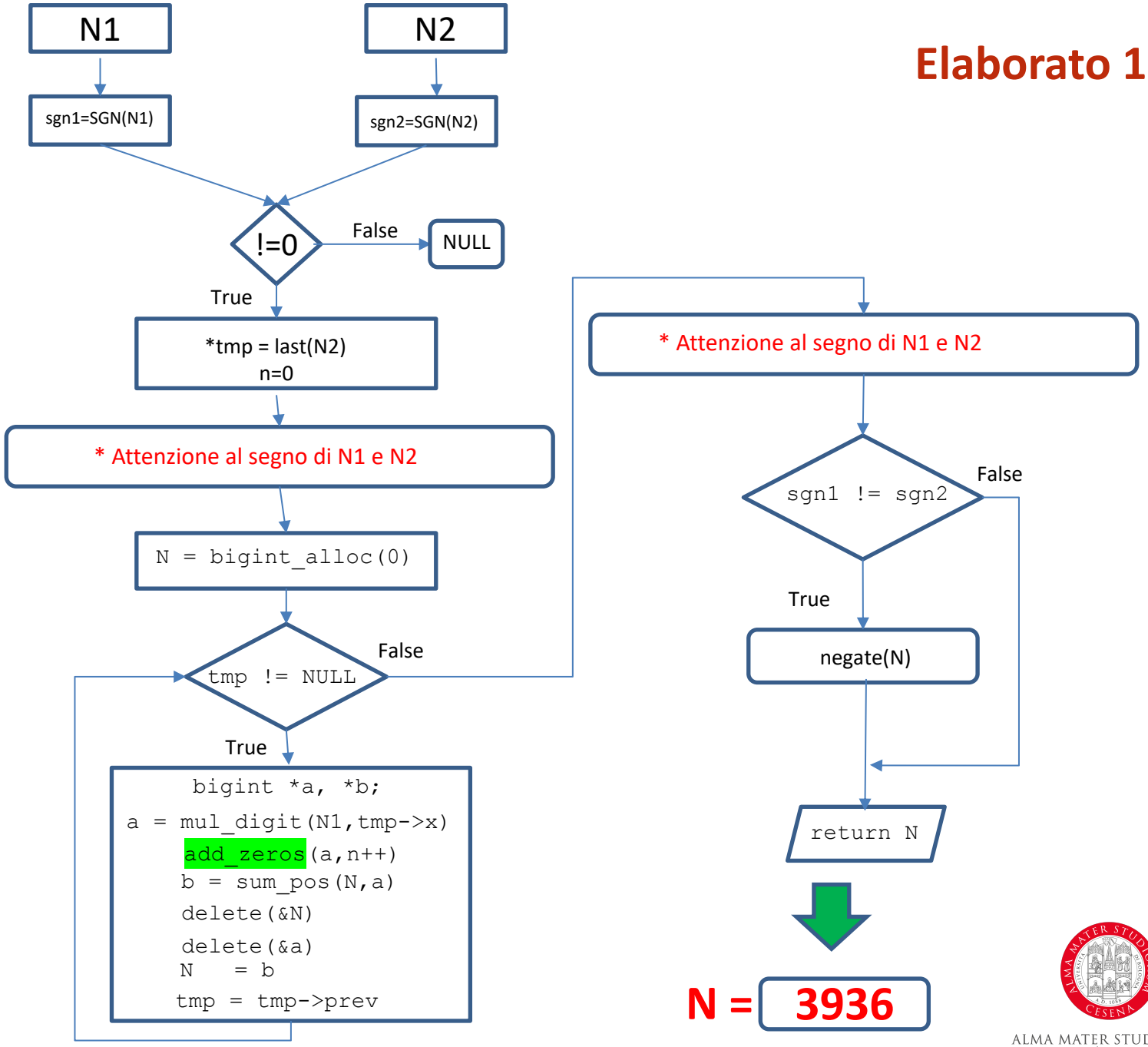
```
static bigint *mul_digit_pos(bigint *N, digit x) {
        bigint *X = NULL;
        int val = 0, car = 0;

  N = last(N);
        while(N != NULL || car != 0) {
                val = (N ? N->x : 0)*x + car;
                car = val / 10;
                val = val % 10;
                head_insert(&X,val);
                N = N ? N->prev : NULL;
        }

        return X;
}
```

Analizziamo le funzioni – head_insert

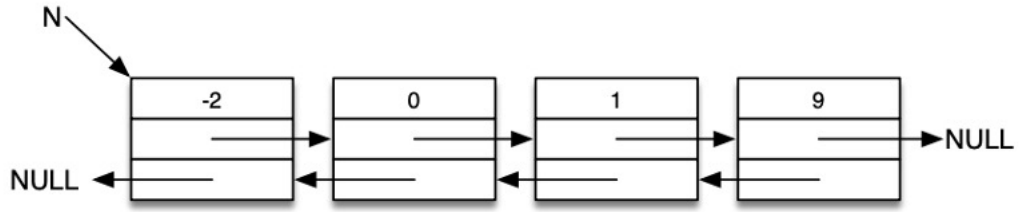



```
static int head_insert(bigint **N, digit x) {
        // se N è NULL{
                    // restituisci 1
        } altrimenti se *N è NULL) {
                restituisci (*N = bigint_alloc(x)) == NULL;
        } altrimenti {
                bigint *tmp = bigint_alloc(x);

                // se tmp è diverso da NULL{
                          // il campo next di tmp è uguale ad *N
                          // il campo prev di(*N) è uguale a tmp
                          // e *N  è uguale a tmp
                }
                return tmp == NULL;
        }
}
```

Elaborato 10

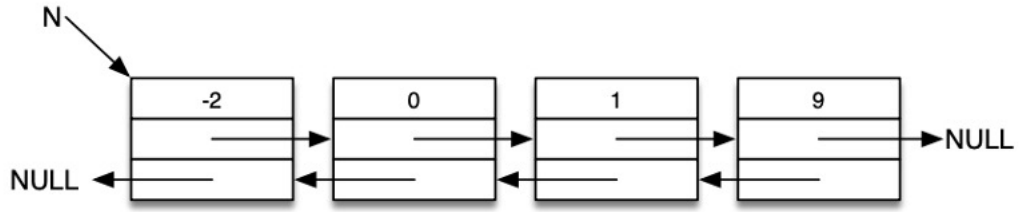


Analizziamo le funzioni – add_zeros



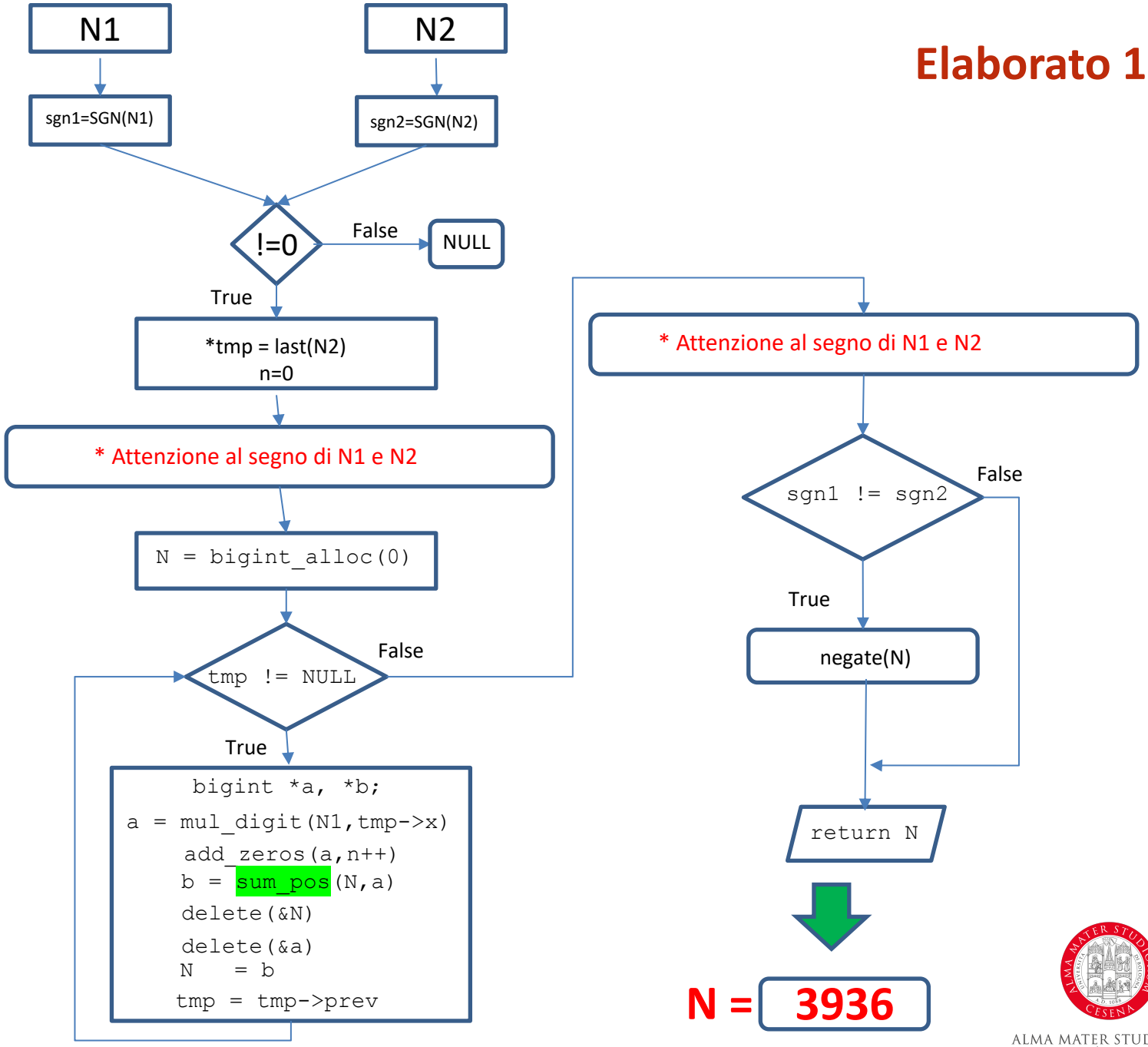

```
static void add_zeros(bigint *N, unsigned int n) {
        // se N è non NULL e il campo x di N è diverso da 0 {
                // definiamo i;
                for(i = 0; i < n; i++)
                        tail_insert(&N,0);
        }
}
```

Analizziamo le funzioni – tail_insert

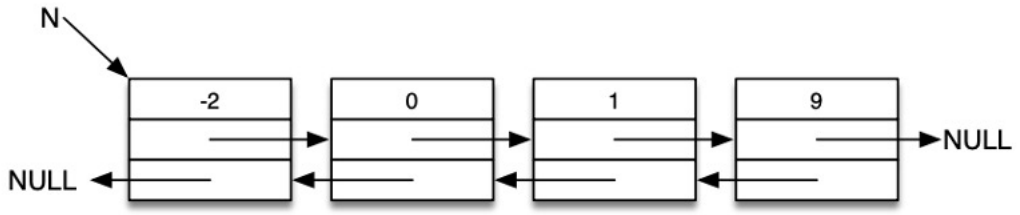



```
static int tail_insert(bigint **N, digit x) {
        // se N è NULL {
        //      restituisci 1;
        } //altrimenti se *N è NULL) {
                restituisci head_insert(N,x);
        } // altrimenti {
                bigint *tmp = last(*N);
                // il campo next di tmp è uguale a bigint_alloc(x);
                // se il campo next di tmp è non NULL
                        il campo prev di tmp->next è uguale a tmp;
                return tmp->next != NULL;
        }
}
```

Elaborato 10

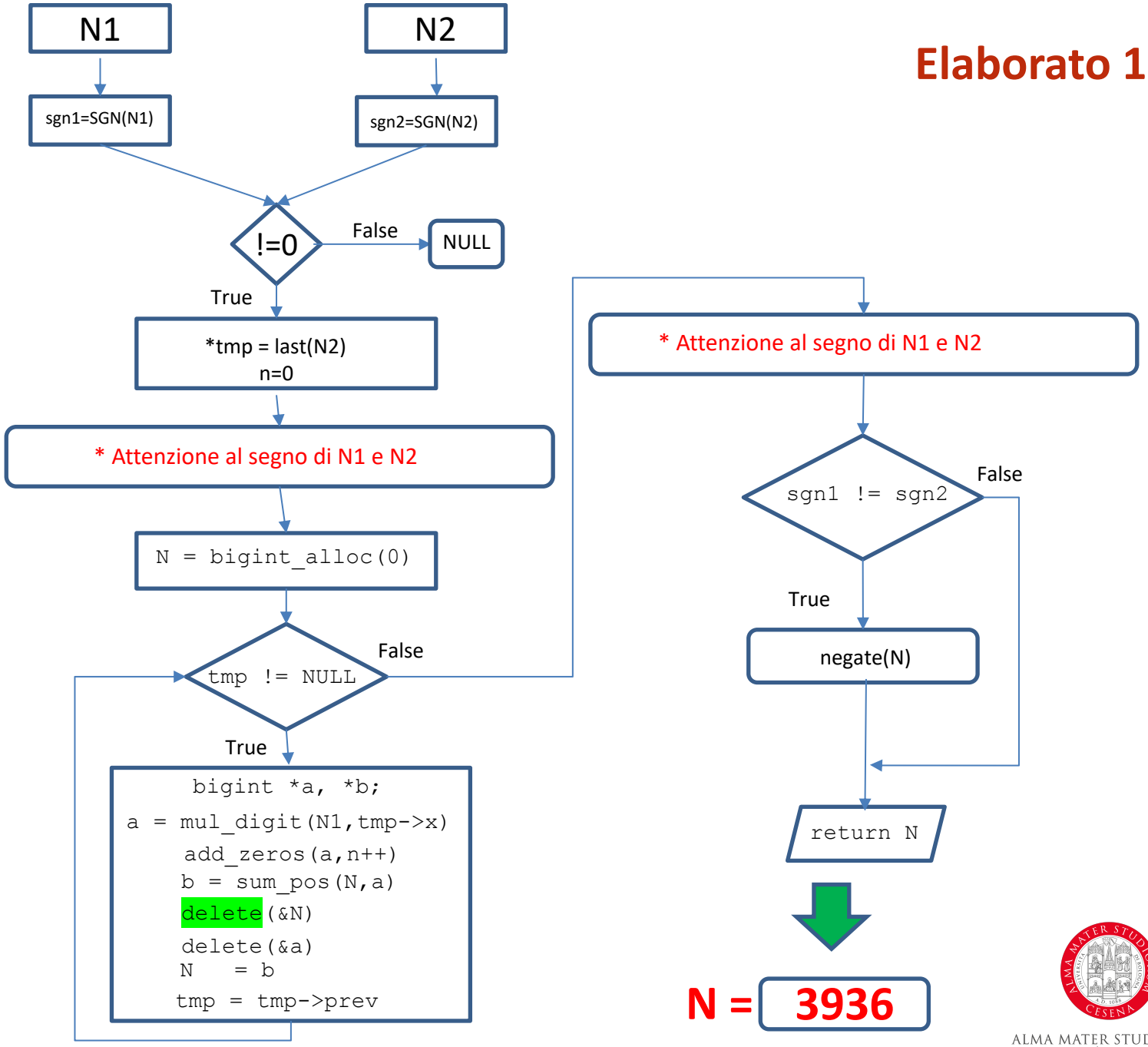


Analizziamo le funzioni – sum_pos

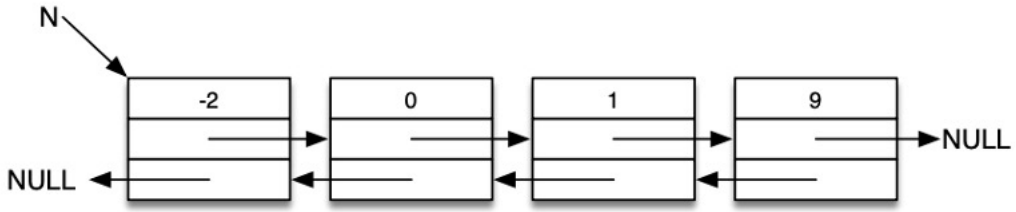



```
static bigint *sum_pos(bigint *N1, bigint *N2) {
        bigint *N = NULL;

        if(SGN(N1) > 0 && SGN(N2) > 0) {
                int val = 0, car = 0;

                N1 = last(N1);
                N2 = last(N2);

                while(N1 != NULL || N2 != NULL || car != 0) {
                        val = (N1 ? N1->x : 0) + (N2 ? N2->x : 0) + car;
                        car = val / 10;
                        val = val % 10;
                        head_insert(&N,val);
                        N1 = N1 ? N1->prev : NULL;
                        N2 = N2 ? N2->prev : NULL;
                }
        }

        return N;
}
```

Elaborato 10

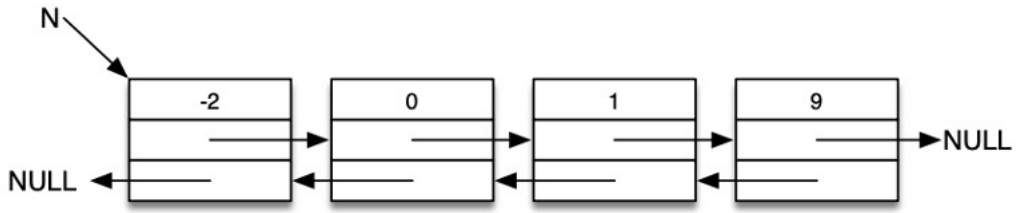


Analizziamo le funzioni – delete



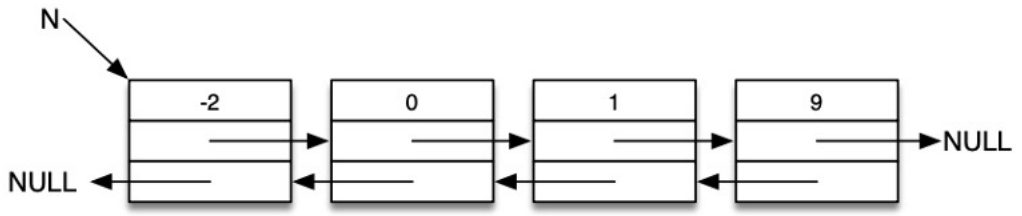

```
static void delete(bigint **N) {
        // se N è non NULL
                // fintanto che *N è non NULL)
                        head_delete(N);
}
```

Analizziamo le funzioni – head_delete



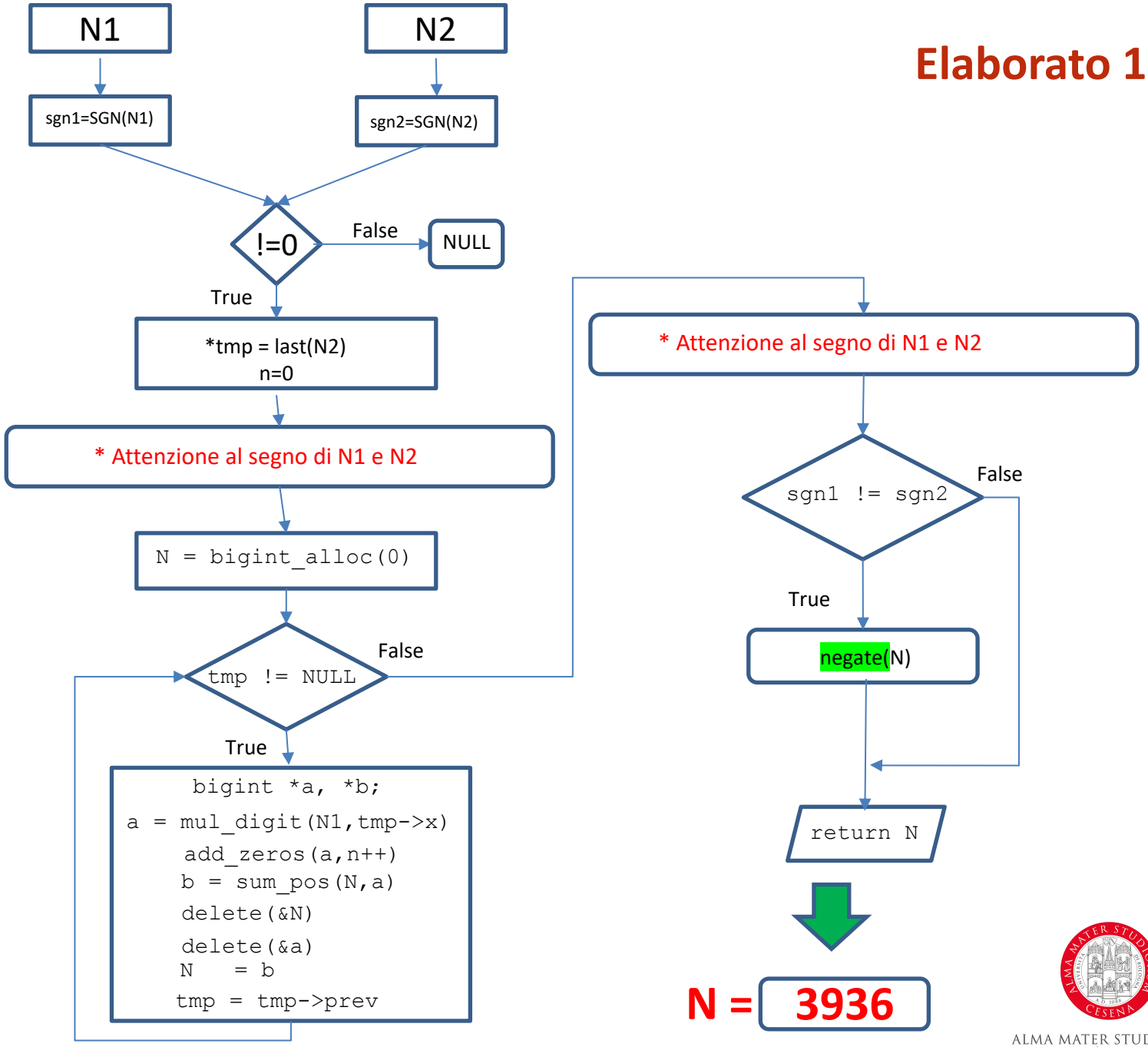

```
static int head_delete(bigint **N) {
        // se N è NULL o *N è NULL {
        //       restituisci 1;
        } //altrimenti {
                bigint *tmp = *N;

                // *N è uguale al campo next di (*N)
                // restituisci bigint_delete(tmp);
        }
}
```

Analizziamo le funzioni – bigint_delete

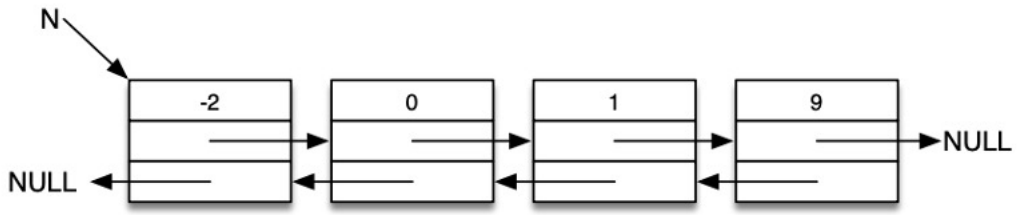



```
static int bigint_delete(bigint *N) {
        // se N è NULL {
                restituisci 1;
        } //altrimenti {
                //se il campo next di N è non NULL
                        // il campo prev di (N->next) è uguale al campo prev di N
                // se il campo prev di N è non NULL
                // il campo next di (N->prev) è uguale al campo next di N
                // libera N;
                // restituisci 0
        }
}
```

Elaborato 10



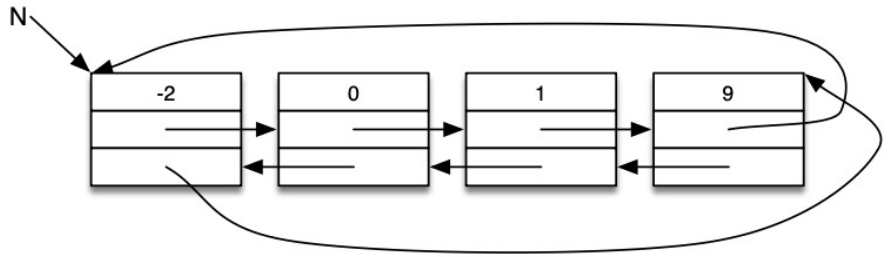
Analizziamo le funzioni – negate




```
static void negate(bigint *N) {
        // se N è non NULL
                // invertire il segno di N
}
```

Elaborato 10

Rappresentazione di un ***bigint*** tramite liste doppiamente concatenate e circolari




```c
typedef signed char digit;

typedef struct bigint {
  digit x;
  struct bigint *next;
  struct bigint *prev;
} bigint;
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA