**ALGORITHMS & DATA STRUCTURES**

ADDITIONAL INFORMATION FOR LAB 17: BIG NUMBERS

WEEK OF 4TH MARCH 2019

1. **PUBLIC VARIABLES**

* ndigits: this variable stores the number of digits of the big integer
* data[]: this array stores the digits of the big integer. The unit is in position 0, the ten is in position 1 and so on. For example, if the (not so) big integer “6283775491” is being processed, then the array data[] should look like this:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 9 | 4 | 5 | 7 | 7 | 3 | 8 | 2 | 6 |
| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] |

Note that this is an array of characters because the only way to represent the initial big number is using a string (that is why the number “6283775491” was written using quotes). If you want to represent a really big number that is bigger than any data type (e.g. “99572613040277179202747592862626194856269384372282922928272645242426278191872626335350”, the only way of doing this is using a string).

1. **CONSTRUCTORS**

C++: **BigInt::BigInt(size\_t \_ndigits)**

Java: **public BigInt(int \_ndigits)**

This constructor initialises the value of the public variable ndigits and allocates memory to the array data[].

C++: **BigInt::BigInt(std::string s)**

Java: **public BigInt(String s)**

This constructor must perform two steps:

1. call the previous constructor to initialise ndigits y allocate memory to data[] (if you don´t take care of this, if only this constructor is called, things are not going to work due to the lack of allocated memory space)
2. copy, digit by digit, the content of string s into data[]

**An important note on storing the digits:**  because the digits are extracted from a string, every one of them is actually a character (char data type) and not a number (integer data type). Remember that characters are 8-bit words. That means that every character is stored as a binary number that has a decimal equivalent. You can check what 8-bit word is used for each character by looking at an ASCII table. For example, if you are going to store the character ‘a’ what is going to be stored is the 8-bit binary number 01100001 (97 in decimal). The following table shows the 8-bit binary numbers used to represent the characters ‘0’, ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’ and ‘9’:

|  |  |  |
| --- | --- | --- |
| **CHARACTER** | **ASCII CODE (binary)** | **ASCII CODE (decimal)** |
| ‘0’ | 0011 0000 | 48 |
| ‘1’ | 0011 0001 | 49 |
| ‘2’ | 0011 0010 | 50 |
| ‘3’ | 0011 0011 | 51 |
| ‘4’ | 0011 0100 | 52 |
| ‘5’ | 0011 0101 | 53 |
| ‘6’ | 0011 0110 | 54 |
| ‘7’ | 0011 0111 | 55 |
| ‘8’ | 0011 1000 | 56 |
| ‘9’ | 0011 1001 | 57 |

For the code of other characters, you can check at [www.asciitable.com](http://www.asciitable.com)

Thus, when you extract digit ‘3’ from the string the number and storing it into data[] you are actually storing the number 51 (the ASCII code of character ‘3’) and not 3. If you do not take any measure regarding this, you are going to get really weird results when you perform arithmetic operations at the digit level. For example if you sum ‘3’ (ASCII code 51) and ‘4’ (ASCII code 52) instead of getting ‘7’ (ASCII code 55) you will get the letter ‘g’ (ASCII code is 103).

Thus, before storing your digits in array data[], you need to “transform” them in the actual numbers you want to use later. So ‘0’ must be stored as 0 and not as 48; ‘1’ must be stored as 1 and not as 49, and so on. Find out what expression is going to help you to do that.

**3. METHODS**

|  |
| --- |
| C++: **unsigned char BigInt::get(size\_t i)**  Java: **public char get(int i)** |

This method receives as an input argument the position of one digit (any number from 0 to ndigits-1). If the input argument is a valid one (a number in the range [0,ndigits)), it then returns the content of array data[] in that position. Otherwise, it returns 0.

|  |
| --- |
| C++: **BigInt \*BigInt::Add(BigInt \*y)**  Java: **public BigInt Add(BigInt y)** |

**C++ CODERS:**

This method receives a pointer to a big integer y (BigInt \*y) as an input argument and returns another pointer to a big integer corresponding to the sum of the numbers represented by y and data[].

Remember that when a variable is created as a pointer (as BigInt \*y) to access its components you have to use the symbol ‘->’ and not the symbol ‘.’ :

* y->ndigits //to get the number of digits of y
* y->data[i] //to access the i-th digit of y
* y->get(i) //to get the i-th digit of y

This function has to:

* Create a pointer to a big integer (e.g. BigInt \*result) where the result of the sum is going to be stored. The number of digits of the result is the maximum number between the number of digits of y and the number of digits of data[]plus 1.
* Perform the sum, digit by digit (sum the i-th digit of y to the i-th digit of data[] plus any carry. For example, if you are summing digits 3 and 9, you are going to get 12. Of course, you should not store 12 in result, because it is a 2-digit number and you only have space for one digit. So, you store the number 2 and take the 1 as a carry for the next single-digit sum. Remember that you are operating with char data types, thus, the carry variable must be a char as well.
* Return the result

**JAVA CODERS:**

This method receives a big integer y (BigInt data type) as an input argument and returns the sum of that big integer (y) and the num

Remember that when a variable is created as a big integer (as BigInt y) to access its components you have to use the symbol ‘.’ :

* y.ndigits //to get the number of digits of y
* y.data[i] //to access the i-th digit of y
* y.get(i) //to get the i-th digit of y

This function has to:

* Create big integer variable (e.g. BigInt result) where the result of the sum is going to be stored. The number of digits of the result is the maximum number between the number of digits of y and the number of digits of data[]plus 1.
* Perform the sum, digit by digit (sum the i-th digit of y to the i-th digit of data[] plus any carry). For example, if you are summing digits 3 and 9, you are going to get 12. Of course, you should not store 12 in result, because it is a 2-digit number and you only have space for one digit. So, you store the number 2 and take the 1 as a carry for the next single-digit sum. Remember that you are operating with char data types, thus, the carry variable and the variable where you store the digit-by-digit sum must be of char data type as well.
* Return the result

|  |
| --- |
| C++: **BigInt \*BigInt::Sub(BigInt \*y)**  Java: **public BigInt Sub(BigInt y)** |

This function returns the result of (data[]-y) (where y is the input argument). You might assume that y is always smaller than or equal to data[].

This method has to:

* Create a variable where to store the result
* Perform the subtraction, digit by digit (the i-th digit of y minus the i-th digit of data[] minus any borrow). For example, if you are calculating 3-5 you are going to get -2. Of course, you should not store -2 in result, because it is a negative number and you only store positive digits. So, you store the number 8 (you borrow one ten for 3 so instead of performing 3-5 you perform 13-5) and take the 1 as a borrow for the next single-digit subtraction. Remember that you are operating with char data types, thus, the borrow variable and the variable where you store the digit-by-digit subtraction must be of char data type as well.
* Return the result

|  |
| --- |
| C++: **BigInt \*BigInt::Shift(size\_t n)**  Java: **public BigInt Shift(int n)** |

* Create a variable where to store the result (with the extra spaces for the extra digits)
* Copy, digit by digit, the digits stored in data[] into their new position (due to the shift)
* Fill the extra spaces in data[] with zeros

|  |
| --- |
| C++: **BigInt \*BigInt::MulByDigit(unsigned char c)**  Java: **public BigInt Shift(int n)** |

This method returns the multiplication of the number stored in datat[] and the single-digit entered as input argument (c).

To implement this method you must:

* Create a big integer where to store the result
* Perform the digit-by-digit multiplication (remember to include the carry bit)
* Return the result

|  |
| --- |
| C++: **BigInt \*BigInt::Mul(BigInt \*y)**  Java: **public BigInt Mul(BigInt y)** |

This method returns the result of multiplying the number stored in data[] with the number entered as an input (y).

To implement this method you must:

* Create a big integer where to store the result
* Use the shift, multiply-by-digit and addition operations to find the result
* Return the result

|  |
| --- |
| C++: **BigInt \*BigInt::Div(BigInt \*y)**  Java: **public BigInt Div(BigInt d)** |

This method returns the result of dividing the number stored in data[] by the number entered as an input (y).

To implement this method you must:

* Create a big integer where to store the result
* Perform the digit by digit division
* Return the result

|  |
| --- |
| C++: **BigInt \*BigInt::Rem(BigInt \*y)**  Java: **public BigInt Mul(BigInt y)** |

This method returns the remainder of the division between the number stored in data[] and the number entered as an input (y).

To implement this method you must:

* Create a big integer where to store the result
* Calculate the reminder
* Return the result