Assignment 1 — Bignum Package

In this assignment you are asked to implement a service that performs basic algebra on integers of arbitrary lengths, called *bignums*. The service includes two main components: a library that provides an API for managing and operating on bignums, and a calculator program that performs basic calculations with arbitrary length integers. Postgraduate students need to implement the complete library. Undergraduates only need to handle positive integers.

1 Library

1.1 Interface

The library provides one abstract data structure `bn_t`, which is a pointer to the (externally abstract) `struct bn`. The functions operating on this structure are described below. This interface is defined in the file `bn.h`, which is available from `myuni`.

1.1.1 `bn_t bn_alloc(void)`

Allocates a new bignum and initialises it to (positive) 0. Returns `NULL` if not enough resources are available. We provide a sample implementation of this function, which you can use.

1.1.2 `void bn_free(bn_t bn)`

Frees all the resources associated with the bignum `bn`.

1.1.3 `int bn_add(bn_t result, bn_t a, bn_t b)`

Adds the bignums `a` and `b` and places the sum in `result`. Returns 0 if completed successfully and -1 otherwise.

1.1.4 `int bn_sub(bn_t result, bn_t a, bn_t b)`

Subtracts the bignum `b` from `a` and places the difference in `result`. Returns 0 if completed successfully and -1 otherwise.

Undergraduate students: if `a` is smaller than `b`, the result is set to 0.

1.1.5 `int bn_mul(bn_t result, bn_t a, bn_t b)`

Multiplies the bignums `a` and `b` and places the product in `result`. Returns 0 if completed successfully and -1 otherwise.

1.1.6 `int bn_fromString(bn_t bn, const char *str)`

Read the decimal number in the string `str` and stores its value in `bn`. Returns 0 if completed successfully and -1 otherwise.

Undergraduate students need only handle non-negative numbers. Postgraduate students need to handle both positive and negative numbers.

1.1.7 `int bn_toString(bn_t bn, char *buf, int buflen)`

Converts a bignum `bn` to a string. including the terminating `NUL` character. If `buflen` is large enough to store the converted string, the function stores the string in `buf` and returns 0. Otherwise, `buf` is not changed and the return value is the number of characters required to store the string representation, The function returns a negative number in case of error.
We provide a sample implementation of this function, which you can use.

1.1.8 `int bn_IAmAnUndergrad()`

Returns 1 if you're an undergraduate student, 0 if you're a postgraduate student. It is highly recommended not to return the wrong value. Implementations that return 0 will be marked on handling both positive and negative numbers.. Implementations that return 1 will only be marked on handling non-negative numbers only, with a penalty of 20% for postgraduate students that use this option.

1.2 Sample Implementation

We provide a very partial implementation of the library. (Available from `myuni`.) This includes a sample declaration of the bignum structure `struct bn` (Figure 1), and implementations of the `bn_alloc` and `bn_toString` function. You do not have to use this implementation, but keep in mind that for Assignment 2 we will extend the sample implementation to also perform division and possibly other operations. If you do not use the sample implementation you will have to implement `bn_toString` and the extended functionality.

```
struct bn {
  int bn_len;
  int bn_size;
  int bn_sign;
  uint16_t *bn_data;
};
```

Figure 1: Bignum abstraction.

The fields of the structure are:

`bn_len` The length (number of digits) of the number represented by the structure.

`bn_data` An array that stores the values of the digits of the number.

`bn_sign` The sign of the number. The value of the sign is 1 for positive numbers and -1 for negative numbers. Undergraduate students should set the field to 1 when allocating numbers and may assume it does not change after that.

`bn_size` The number of digits allocated in `bn_data`.

The number is reprsented as a sequence of *digits* in base $2^{16} = 65536$. That is, if s is the value of `bn_sign`, l the value of `bn_len`, and d_i the value of the i^{th} element in `bn_data` (i.e. `bn_data[i]`), then the value represented by the structure is given by: $s \cdot \sum_{i=0}^{l-1} (2^{16})^i d_i$.

1.3 An Example

Figure 2 shows a program that uses the library to calculate the first 1000 elements of the Fibonacci sequence.

```c
#include <stdio.h>
#include "bn.h"

int main() {
  char buf[1000];
  bn_t a = bn_alloc();
  bn_t b = bn_alloc();

  bn_fromString(a, "0");
  bn_fromString(b, "1");

  for (int i = 0; i < 1000; i++) {
    bn_toString(a, buf, sizeof(buf));
    printf("%2d: %s\n", i, buf);
    bn_add(a, a, b);
    bn_t t = a;
    a = b;
    b = t;
  }
```

Figure 2: Calculating the first 1000 Fibonacci numbers.

2 Calculator

Word	**Description**
Decimal number	Decimal numbers consist of a sequence of one or more decimal digits (`'0'`–`'9'`). For postgraduate students, the number may be preceded by a sign (`'-'` or `'+'`). When processed, the input number is converted into a bignum which is pushed to the stack.
`+`, `-`, or `*`	Pops two values from the stack and pushes their sum, difference, or product back to the stack. in the case of subtraction, the bignum at the stack top is subtracted from the value below it. For example, the sequence `"5 3 -"` leaves the number 2 at the top of the stack.
`dup`	Copies the value of the bignum at the top of the stack and pushes the duplicate value into the stack.
`pop`	Pops a bignum from the top of the stack.
`print`	Prints the value of the bignum at the top of the stack followed by a newline (`'\n'`).
`swap`	Swaps the order of the two bignums at the top of the stack.
`dump`	Prints the contents of the stack, one bignum per line, starting from the stack top. The stack is not modified.
`clear`	Clears the stack.

Table 1: Calculator syntax.

The calculator uses the library you developed to evaluate arithmetic expressions and display the results. The input consists of a sequence of *words* separated by one or more white spaces. (Technically, a white space character is a character for which `isspace()` returns `TRUE`.) Table 1 summarises the words that the calculator recognises.

2.1 Error Handling

In case of error, your calculator should issue an error message. The message should be printed to `stderr` and should start with the word 'Error'.

2.2 Examples

Input	Output
`3 5 + print`	`8`
`3` `5 + print`	 `8`

`3 dup * print dup * print dup * print`	`9`
	`81`
	`6561`

`3 5 - print`	`0` (undergraduates) `-2` (postgraduates)

3 Submission Guideline

You should submit a `tar` or a `tgz` file. When the contents of the file is extracted, it creates a single folder whose name is your a number. The folder should contain a `Makefile` and all of the sources required for building your library and calculator. Running `make` in the folder should create the library `libbn.a` and the binary `calc`. It may also create other intermediate files, e.g. object files.

Make should compile all the sources with `-Wall`, and the compilation should issue no warnings. (Note that you may only use C for the assignment).

We will provide a test program that you can use to perform some sanity tests on your submission.