Using finite state techniques design a lexical analyser for processing a sequence of 32-bit octal, hexadecimal and signed integer constants (lexemes) described by the following regular expression:

```
[0-7]+[bB] | [0-9a-fA-F]+[hH] | ((+|—)?[0-9]+)
```

Code the lexical analyser in either C or C++ and verify that it works correctly by writing a test program to read a sequence of lexemes from the standard input, printing each one recognized on the standard output together with a description of its corresponding lexical token. Any invalid lexemes should be flagged.

For example, given the input sequence:

4096 4321056b 4321056h 123456789h

the output might look like:

Lexeme “4096”	Lexical token (constant, 4096)
Lexeme “4321056b”	Lexical token (constant, 1155630)
Lexeme “804020C0h”	Lexical token (constant, -2143280960)
Lexeme “123456789h”	Error, constant contains too many digits

Note:

i. Octal constants have base “`b`” whereas hexadecimal constants have base “`h`”.

ii. The largest permissible positive octal and hexadecimal constants are “`17777777777b`” and “`7FFFFFFFh`” respectively—both of these values represent the constant 2147483647.

iii. A complete set of test inputs designed to visit all non-error entries in your transition table should be included with the design description.

iv. Great care should be taken to avoid arithmetic overflow when processing values around either end of the permissible range—ie -2^{31} .. $2^{31}-1$.

v. All processing operations in the analyser are to be performed using 32-bit arithmetic.