Please notice that I also implement multiply operation.

1. Structure

Define a structure named number to store inputted number

```
typedef struct number_ {
    char type;
    char *numString;
    int negative;
    int equiv;
} number;
```

2. Functions

int isDigit (char x);
Check if a character is a digit
int Pow (int power, int base);
Comput the power of a int (To avoid the math lib)
int isBinary(char* numB);
Check if a String is in correct binary format
int isOctal(char* numO);
Check if a String is in correct Octal format
int isHex (char* numH);
Check if a String is in correct Hex format
int isDecimal(char* numD);
Check if a String is in correct Decimal format
int validateToken(char* num);
Check if a String is a valid token or not
int strToInt(number *ptr);
Convert a String (From number structure) to Int
int hexToInt(number* ptr);
Convert a hex (From number structure) to Int
int convertToInt(number* ptr);
Convert a String (From number structure) to Int
number* numCreate (char* str);
Create a number structure from a String
char* toStr(char type, int ans);
Convert a int, octal, hex to Int
char* toHex(int ans);

Convert the answer to Hex
char* convertAns (char type, int ans);
Convert the answer to a certain tyoe (except for Hex)
int checkResult(char type, int ans, number* num1, number* num2);
Check if the output result is 32 bits or not
int Result(char op, number* num1, number* num2);
compute the result of * + - operations
void delete (number* num1, number* num2, char* str);
Free all the tokens
3. The most challenge part is to avoid the math lib for me.
4. Best Case: 2 32 bit int add up, output is in int form and the result is also within 32 bit.
Big (O) = Big(1)
Worst Case: two Binary add up, output is in also in binary form.