Peanut Extension Module - BNA

Author: farmerliao.tw@gmail.com

This extension module for Peanut is to provide big number arithmetic (bna) functions on Peanut.

Like many computer languages, number of bits of integers is limited in Peanut, too. This extension for Peanut provides **integer** calculations with unlimited digits.

In this extension, each big number is represented by a handle in Peanut and we call it a bigNum handle. A bigNum handle is returned from bna extension functions after the processing. As any big number is not used any more, the script should call bna function(s) to recycle the memory resource used for that big number.

Namespace

The namespace bna is created by this extension. All the extension functions registered to Peanut can be only called with namespace resolution `bna::` preceded.

Nibble

A big number might be thousands of digits. In the internals of the bna program, a big number is decomposed into nibbles. Each nibble is a fixed-size unsigned integer. The nibble size can be read from the constant bna::nibbleSize, which denotes the number of bits of a nibble. And the values of the nibbles can be read by specifying the index of the nibble. Say that one bigNum has 4 nibbles (refer to the following for an example); the indexes for accessing the nibble values are from 0 to 3; the least significant nibble is at index 0 and get the nibble value 0x02345678; the most significant nibble is at index 3 and the nibble value is 0x32345678.

Index	0	1	2	3
	02345678	12345678	22345678	32345678

In the following paragraphs, parameters and output values of extension functions are described.

Extension Functions

Prototype	`new(spec)`	
Description	Generate a big number.	
Return Value	A bigNum handle	
Parameter	spec	Either an integer or a string can be specified for bna to generate a bigNum handle. Integer: If an integer is used as the input, one should note that Peanut only support integers up to either 32 bits or 64 bits long depending on the number of bits of a Peanut integer. String: If a string is used as the input to generate a bigNum, as many as possible digits can be specified. If a string is used to specify the bigNum value, the following shows the formats. Any comma characters are skip. User can use comma characters in the string to improve the readability. “123456789123456789123456789” (in decimal) “0x123456789ABCDE123456789ABCDE” (in hexadecimal) “- 123456789” (negative; in decimal) “- 0x123456789” (negative; in hexadecimal) “123,456,789,123,456,789,123,456,789”
See Also		

```
#load <pxm_bna.pxm>

main()
{
    var n1, n2, n3;

    n1 = bna::new(123);
    n2 = bna::new("456");
    n3 = bna::new("1234567890123456789012345678901234567890123456789 0");
    bna::del(n1, n2, n3);
}
```

Prototype	`clone(num)`	
Description	Glone a big number.	
Return Value	A bigNum handle	
Parameter	num	A bigNum handle
See Also		

```
#load <pxm_bna.pxm>

main()
{
    var n1, n2;

    n1 = bna::new("1234567890123456789012345678901234567890123456789O");
    n2 = bna::clone(n1);
    bna::dump10(n2);

    bna::del(n1, n2);
}
```

Prototype	`del(num, ...)`	
Description	Delete one or multiple big numbers. This is done while the big number is not needed any more. The system recycles the resource used for this big number.	
Return Value	0	
Parameter	num	A bigNum handle
	…	Optional bigNum handles
See Also		

```
#load <pxm_bna.pxm>

main()
{
    var n1, n2, n3;

    n1 = bna::new(123);
    n2 = bna::new(456);
    n3 = bna::exp(n1, n2);
    bna::del(n1, n2, n3);
}
```

Prototype	`nibble(num, idx)`	
Description	Get the nibble value at the specified index.	
Return Value	An unsigned integer indicating the nibble value at the specified index of the bigNum num.	
Parameter	num	A bigNum handle
	Idx	The index of the nibble
See Also	bna::nibbleSize bna::nibbleNum()	

```
#load <pxm_bna.pxm>

main()
{
    var n1, n2, n3;
    var nbnum, MSNibble;

    n1 = bna::new(123);
    n2 = bna::new(456);
    n3 = bna::exp(n1, n2);
    bna::del(n1);
    bna::del(n2);
    nbnum = bna::nibbleNum(n3);
    MSNibble = bna::nibble(n3, nbnum - 1);
    print("MSNibble = ", MSNibble);

    bna::del(n1, n2, n3);
}
```

Prototype	`nibbleNum(num)`	
Description	Get the number of nibbles of the bigNum.	
Return Value	An unsigned integer indicating the number of nibbles of the specified bigNum num.	
Parameter	num	A bigNum handle
See Also	bna::nibbleSize bna::nibble()	

```
#load <pxm_bna.pxm>

main()
{
    var n1, n2, n3;
    var nbnum, MSNibble;

    n1 = bna::new(123);
    n2 = bna::new(456);
    n3 = bna::exp(n1, n2);
    bna::del(n1);
    bna::del(n2);
    nbnum = bna::nibbleNum(n3);
    MSNibble = bna::nibble(n3, nbnum - 1);
    print("MSNibble = ", MSNibble);

    bna::del(n1, n2, n3);
}
```

Prototype	`cmp(num1, num2)`	
Description	Compare values of two bigNum numbers.	
Return Value	-1: num1 is less than num2 0 : num1 is equal to num2 1 : num1 is greater than num2	
Parameter	num1	The handle of the first bigNum to be compared
	num2	The handle of the second bigNum to be compared
See Also	bna::nibbleSize bna::nibble()	

```
#load <pxm_bna.pxm>

main()
{
    var n1, n2, n3, n4;
    var r;

    n1 = bna::new(123);
    n2 = bna::new(456);
    n3 = bna::exp(n1, n2);
    n4 = bna::exp(n2, n1);
    r = bna::cmp(n3, n4);
    if (r == 0)
        print("123^456 = 456^123");
    else if (r == 1)
        print("123^456 > 456^123");
    else
        print("123^456 < 456^123");

    bna::del(n1, n2, n3, n4);
}
```

Prototype	`dump10(num)`	
Description	Print the value of the specified bigNum number in decimal format.	
Return Value	0	
Parameter	num	The handle of the bigNum
See Also	dump16()	

```
#load <pxm_bna.pxm>

main()
{
    var n1, n2, n3;

    n1 = bna::new(123);
    n2 = bna::new(456);
    n3 = bna::exp(n1, n2);
    print("123^456 = "); bna::dump10(n3);

    bna::del(n1, n2, n3);
}
```

Prototype	`dump16(num)`	
Description	Print the value of the specified bigNum number in hexadecimal format.	
Return Value	0	
Parameter	num	The handle of the bigNum
See Also	dump10()	

```
#load <pxm_bna.pxm>

main()
{
    var n1, n2, n3;

    n1 = bna::new(123);
    n2 = bna::new(456);
    n3 = bna::exp(n1, n2);
    print("123^456 = "); bna::dump16(n3);

    bna::del(n1, n2, n3);
}
```

Prototype	`neg(num)`	
Description	Multiply the specified number with integer -1.	
Return Value	A bigNum handle	
Parameter	num	handle of a bigNum
See Also	mul	

```
#load <pxm_bna.pxm>

main()
{
    var n1, n2, n3, n4;

    n1 = bna::new(123);
    n2 = bna::new(456);
    n3 = bna::exp(n1, n2);
    n4 = bna::neg(n3);
    print("n4 = "); bna::dump10(n4);

    bna::del(n1, n2, n3, n4);
}
```

Prototype	`add(num1, num2)`	
Description	Calculate num1 + num2	
Return Value	The sum of the two numbers.	
Parameter	num1	handle of the first operand of addition
	num2	handle of the second operand of addition
See Also		

```
#load <pxm_bna.pxm>

main()
{
    var n1, n2, fnew;
    var i;

    n1 = bna::new(0);
    n2 = bna::new(1);
    print("f(1) = "); bna::dump10(n1); print("\n");
    print("f(2) = "); bna::dump10(n2); print("\n");
    for (i = 3; i <= 1000; ++i)
    {
        fnew = bna::add(n1, n2);
        print("f(", i, ") = "); bna::dump10(fnew); print("\n");
        bna::del(n1);
        n1 = n2;
        n2 = fnew;
    }
    bna::del(n1, n2);
}
```

Prototype	`modAdd(num1, num2, m)`	
Description	Do a modular addition calculation. That is to calculate num1 + num2 mod m.	
Return Value	A bigNum handle of the calculation result.	
Parameter	num1	handle of the first operand of modular addition
	num2	handle of the second operand of modular addition
	m	handle of the third operand of the modular addition; that is the modulus
See Also		

Prototype	`sub(num1, num2)`	
Description	Calculate num1 - num2	
Return Value	A bigNum handle of the calculation result.	
Parameter	num1	handle of the first operand of subtraction
	num2	handle of the second operand of subtraction
See Also		

Prototype	`modSub(num1, num2, m)`	
Description	Do a modular subtraction calculation. That is to calculate num1 - num2 mod m	
Return Value	A bigNum handle of the calculation result.	
Parameter	num1	handle of the first operand of modular subtraction
	num2	handle of the second operand of modular subtraction
	m	handle of the third operand of modular subtraction; that is the modulus
See Also		

Prototype	`mul(num1, num2)`	
Description	Calculate num1 * num2	
Return Value	A bigNum handle of the calculation result.	
Parameter	num1	handle of the first operand of multiplication
	num2	handle of the second operand of multiplication
See Also		

Prototype	`modMul(num1, num2, m)`	
Description	Do a modular multiplication calculation. That is to calculate num1 * num2 mod m.	
Return Value	A bigNum handle of the calculation result.	
Parameter	num1	handle of the first operand of modular multiplication
	num2	handle of the second operand of modular multiplication
	m	handle of the third operand of modular multiplication; that is the modulus
See Also		

Prototype	`div1(num1, num2, rem)`	
Description	Do an integer division calculation. That is to calculate num1 / num2.	
Return Value	A bigNum handle of the quotient	
Parameter	num1	handle of the dividend
	num2	handle of the divisor
	rem	This parameter is a **call-by-reference parameter**. A variable should be passed for the div1 function to store the value of the remainder.
See Also		

```
#load <pxm_bna.pxm>

main()
{
    var dividend, divisor, Q, R, P;
    var temp;

    dividend = bna::new("100,000,000,000,000,000,000,000,000,000,000");
    divisor  = bna::new("987654321987654321987654321");
    Q = bna::div1(dividend, divisor, R);

    print("Q = "); bna::dump10(Q);
    print("R = "); bna::dump10(R);

    temp = bna::mul(divisor, Q);
    P = bna::add(temp, R);
    print("P = "); bna::dump10(P);

    bna::del(dividend, divisor, Q, R, temp, P);
}
```

Prototype	`mod(num, m)`	
Description	Do moduo operation num mod m; or say do an integer division calculation and get the remainder.	
Return Value	A bigNum handle of the remainder	
Parameter	num	A bigNum handle
	m	handle of the modulus
See Also		

```
#load <pxm_bna.pxm>

main()
{
    var dividend, divisor, Q, R1, R2;

    dividend = bna::new("100,000,000,000,000,000,000,000,000,000,000");
    divisor  = bna::new("987654321987654321987654321");
    R1 = bna::mod(dividend, divisor);
    Q  = bna::div1(dividend, divisor, R2);
    if (bna::cmp(R1, R2) == 0)
        print("Correct!");
    else
        print("Wrong!");
    bna::del(dividend);
    bna::del(divisor);
    bna::del(Q);
    bna::del(R1);
    bna::del(R2);
}
```

Prototype	`exp(base, power)`	
Description	Do exponentiation.	
Return Value	A bigNum handle of the calculation result.	
Parameter	base	handle of the base number
	power	handle of the power number
See Also		

```
#load <pxm_bna.pxm>

main()
{
    var n1, n2, n3;

    n1 = bna::new(123);
    n2 = bna::new(456);
    n3 = bna::exp(n1, n2);
    print("123^456 = "); bna::dump10(n3);

    bna::del(n1, n2, n3);
}
```

Prototype	`modExp(base, power, m)`	
Description	Calculate exponentiation over a modulus.	
Return Value	A bigNum handle of the calculation result.	
Parameter	base	handle of the base number
	power	handle of the power number
	m	handle of the modulus
See Also		

```
#load <pxm_bna.pxm>

main()
{
    var n1, n2, n3;

    n1 = bna::new(1234);
    n2 = bna::new(53117);
    n3 = bna::modExp(n1, n2, n2);
    print("1234^53117 mod 53117 = "); bna::dump10(n3);

    bna::del(n1, n2, n3);
}
```

Prototype	`gcd(num1, num2)`	
Description	Calculate the greatest common divisor.	
Return Value	A bigNum handle of the calculation result.	
Parameter	num1	handle of the first number
	num2	handle of the second number
See Also		

```
#load <pxm_bna.pxm>

main()
{
    var n1, n2, n3;

    n1 = bna::new("12340000000000000000000");
    n2 = bna::new("53117000000000000000000");
    n3 = bna::gcd(n1, n2);
    bna::dump10(n3);
    bna::del(n1, n2, n3);
}
```

Prototype	`modMulInv(num, m)`	
Description	Calculate the modular multiplication inverse. That is to find the interger x , where x < m, such that 1 = num * x mod m.	
Return Value	A bigNum handle of the calculation result. If the handle value is 0, there exists no multiplication inverse of num over modulus m.	
Parameter	num	handle of the number whose multiplication inverse would be calculated
	m	handle of the modulus
See Also		

```
#load <pxm_bna.pxm>

main()
{
    var n1, n2, n3, n4;
    var constant_1;

    constant_1 = bna::new(1);
    n1 = bna::new(1234);
    n2 = bna::new(53117);
    n3 = bna::modMulInv(n1, n2);
    n4 = bna::modMul(n1, n3, n2);

    if (bna::cmp(n4, constant_1) != 0)
        print("Wrong answer is got: ");
    else
        print("Multiplication inverse = ");
    bna::dump10(n3);

    bna::del(n1, n2, n3, n4);
    bna::del(constant_1);
}
```

Prototype	`lshift(& num, n)`	
Description	Left shift (up shift) the bigNum num with n bits.	
Return Value	0	
Parameter	num	handle of the number to be shifted; this parameter is a call-by-reference one
	n	a positive integer
See Also		

Prototype	`eccFpNewCurve(p, n, a, b, Gx, Gy)`	
Description	Create an elliptic curve with coefficients a and b over the finite field *F*p, where (Gx, Gy) is the selected base point in *F*p X *F*p. The elliptic equation `y^2 = x^3 + ax + b mod p …(1)`	
Return Value	handle of the created elliptic curve	
Parameter	p	a handle of a prime number specifying *F*p
	n	a handle of the order of base point
	a	a handle of the coefficient a in equation (1)
	b	a handle of the coefficient b in equation (1)
	Gx	a handle of the x-coordinate value of the base point
	Gy	a handle of the y-coordinate value of the base point
See Also		

```
#load <pxm_bna.pxm>

read_SECP384r1_params(& nKeyLenInBit,
                      & p,
                      & n,
                      & a,
                      & b,
                      & Gx,
                      & Gy)
{
// SECP384r1

    var secp384r1ParamString_p;
    var secp384r1ParamString_n;
    var secp384r1ParamString_a;
    var secp384r1ParamString_b;
    var secp384r1ParamString_Gx;
    var secp384r1ParamString_Gy;



secp384r1ParamString_p =
// ruler
//                  1       2       3       4       4       5       6       7       8       8       9
//   1      8       6       4       2       0       8       6       4       2       0       8       6
  "0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFF0000000000000000FFFFFFFF";

secp384r1ParamString_n =
// ruler
//                  1       2       3       4       4       5       6       7       8       8       9
//   1      8       6       4       2       0       8       6       4       2       0       8       6
  "0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC7634D81F4372DDF581A0DB248B0A77AECEC196ACCC52973";

secp384r1ParamString_a =
// ruler
//                  1       2       3       4       4       5       6       7       8       8       9
//   1      8       6       4       2       0       8       6       4       2       0       8       6
  "0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFF0000000000000000FFFFFFFC";

secp384r1ParamString_b =
// ruler
//                  1       2       3       4       4       5       6       7       8       8       9
//   1      8       6       4       2       0       8       6       4       2       0       8       6
  "0xB3312FA7E23EE7E4988E056BE3F82D19181D9C6EFE8141120314088F5013875AC656398D8A2ED19D2A85C8EDD3EC2AEF";

secp384r1ParamString_Gx =
// ruler
//                  1       2       3       4       4       5       6       7       8       8       9
//   1      8       6       4       2       0       8       6       4       2       0       8       6
  "0xAA87CA22BE8B05378EB1C71EF320AD746E1D3B628BA79B9859F741E082542A385502F25DBF55296C3A545E3872760AB7";

secp384r1ParamString_Gy =
// ruler
//                  1       2       3       4       4       5       6       7       8       8       9
//   1      8       6       4       2       0       8       6       4       2       0       8       6
  "0x3617DE4A96262C6F5D9E98BF9292DC29F8F41DBD289A147CE9DA3113B5F0B8C00A60B1CE1D7E819D7A431D7C90EA0E5F";


    nKeyLenInBit = 384;
    p  = bna::new(secp384r1ParamString_p);
    n  = bna::new(secp384r1ParamString_n);
    a  = bna::new(secp384r1ParamString_a);
    b  = bna::new(secp384r1ParamString_b);
    Gx = bna::new(secp384r1ParamString_Gx);
    Gy = bna::new(secp384r1ParamString_Gy);
}

main()
{
    var nKeyLen, p, n, a, b, Gx, Gy;
    var anECC;
    var k1, k2, k3;
    var Px, Py;
    var Qx, Qy;
    var Rx, Ry;
    var Points[3][2];

    read_SECP384r1_params(nKeyLen, p, n, a, b, Gx, Gy);

    k1 = bna::new("97699997777666555332211456781212128989");
    k2 = bna::new("12345678987654321234567898765432123456789");
    k3 = bna::mul(k1, k2);

    anECC = bna::eccFpNewCurve(nKeyLen, p, n, a, b, Gx, Gy);

    //(Px, Py) = k1 * (Gx, Gy)
    bna::eccFpPointMul(anECC, Gx, Gy, k1, Px, Py);

    //(Qx, Qy) = n  * (Px, Py)
    bna::eccFpPointMul(anECC, Px, Py, k2, Qx, Qy);

    //(Rx, Ry) = k2 * (Gx, Gy)
    bna::eccFpPointMul(anECC, Gx, Gy, k3, Rx, Ry);

    print("Qx = "), bna::dump16(Qx);
    print("Qy = "), bna::dump16(Qy);
    print("Rx = "), bna::dump16(Rx);
    print("Ry = "), bna::dump16(Ry);

    bna::eccFpPointMul(anECC, Gx, Gy, k1, Points[0][0], Points[0][1]);
    bna::eccFpPointMul(anECC, Points[0][0], Points[0][1], k2, Points[1][0], Points[1][1]);
    bna::eccFpPointMul(anECC, Gx, Gy, k3, Points[2][0], Points[2][1]);
    print("Points[1][0] = "), bna::dump10(Points[1][0]);
    print("Points[1][1] = "), bna::dump10(Points[1][1]);
    print("Points[2][0] = "), bna::dump10(Points[2][0]);
    print("Points[2][1] = "), bna::dump10(Points[2][1]);

    bna::del(Px, Py);
    bna::del(Qx, Qy);
    bna::del(Rx, Ry);

    for ($1 = 0; $1 < 3; ++$1)
    for ($2 = 0; $2 < 2; ++$2)
        bna::del(Points[$1][$2]);

    bna::eccFpDelCurve(anECC);

}
```

Prototype	`eccFpDelCurve(curve)`	
Description	Delete the elliptic curve bigNum handle. That is to recycle the resource used for this curve handle.	
Return Value	0	
Parameter	curve	handle of elliptic curve returned by `eccFpNewCurve` function
See Also	`eccFpNewCurve`	

Prototype	`eccFpPointMul(curve, Px, Py, k, &Qx, &Qy)`	
Description	Do an elliptic curve point multiplication calculation. That is to calculate Qx and Qy such that (Qx, Qy) = k * (Px, Py) on curve. (Q = kP on curve)	
Return Value	0	
Parameter	curve	handle of elliptic curve returned by `eccFpNewCurve` function
	Px	The handle of the x-coordinate value of point P.
	Py	The handle of the y-coordinate value of point P.
	k	An integer
	Qx	A variable for this function to update bigNum handle of the x-coordinate value (the handle of the x-coordinate value of point Q)
	Qy	A variable for this function to update bigNum handle of the y-coordinate value (the handle of the y-coordinate value of point Q)
See Also	`eccFpNewCurve`	