9 C Language Function Block

In this chapter, we focus on C language function block's specifications, edition, instruction calling, application points etc. we also attach the common Function list.

9-1. Summary

This is the new added function in XCPPro software. This function enable the customers to write function blocks with C language in XCPPo; and call the function blocks at any necessary place. This function supports most of C language functions, strength the program's security. As users can call the function at many places and call different functions, this function increase the programmer's efficiency greatly.

9-2. Instruction Format

1、Instruction Summary

Call the C language Func Block at the specified place

Call the C language Func Block [NAME_C]			
16 bits Instruction	NAME_C	32 bits Instruction	-
Execution Condition	Normally ON/OFF, Rising/Falling Edge activation	Suitable Models	XC1、XC2、XC3、XC5、XCM
Hardware Requirement	V3.0C and above	Software Requirement	V3.0C and above

2、Operands

Operands	Function	Type
S1	name of C Func Block, defined by the user	String
S2	Correspond with the start ID of word **W** in C language Function	16bits, BIN
S3	Correspond with the start ID of word **B** in C language Function	16bits, BIN

3、Suitable Soft Components

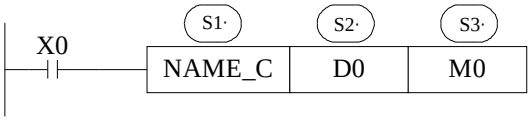
Word

Operands	System									Constant	Module	
	D	FD	ED	TD	CD	DX	DY	DM	DS	K/H	ID	QD
S2	•											

Bit

Operands	System						
	X	Y	M	S	T	C	Dn.m
S3			•				

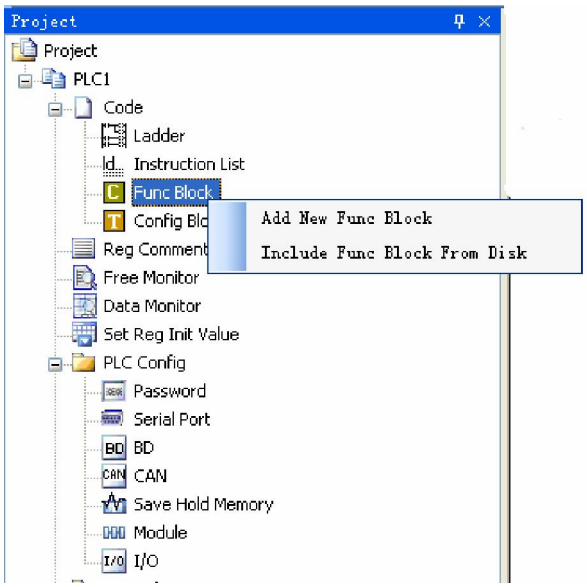
Functions and Actions



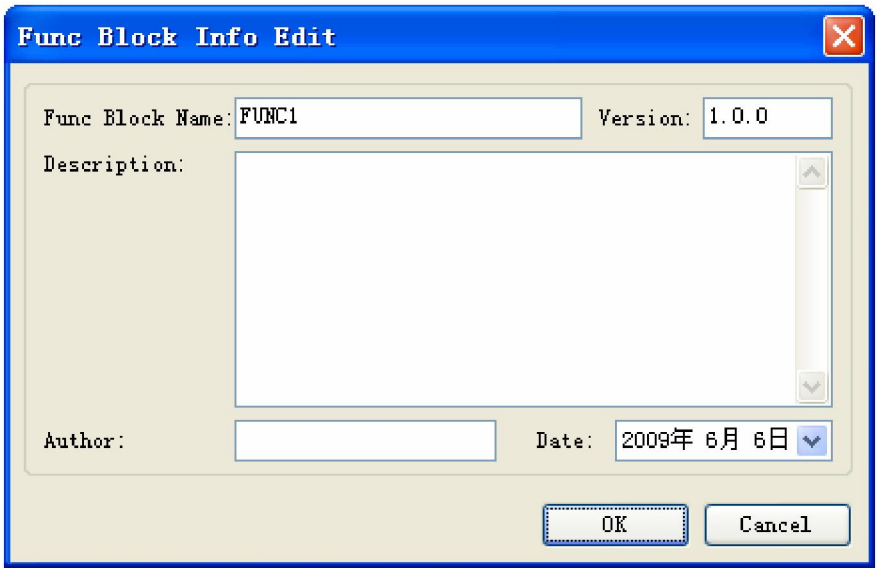
- The name is composed by numbers, letters and underlines, the first character can't be numbers, the name's length shouldn't longer than 8 ASC.
- The name can't be same with PLC's self instructions like LD,ADD,SUB,PLSR etc.
- The name can't be same with the func blocks exist in current PLC;

9-3. Operation Steps

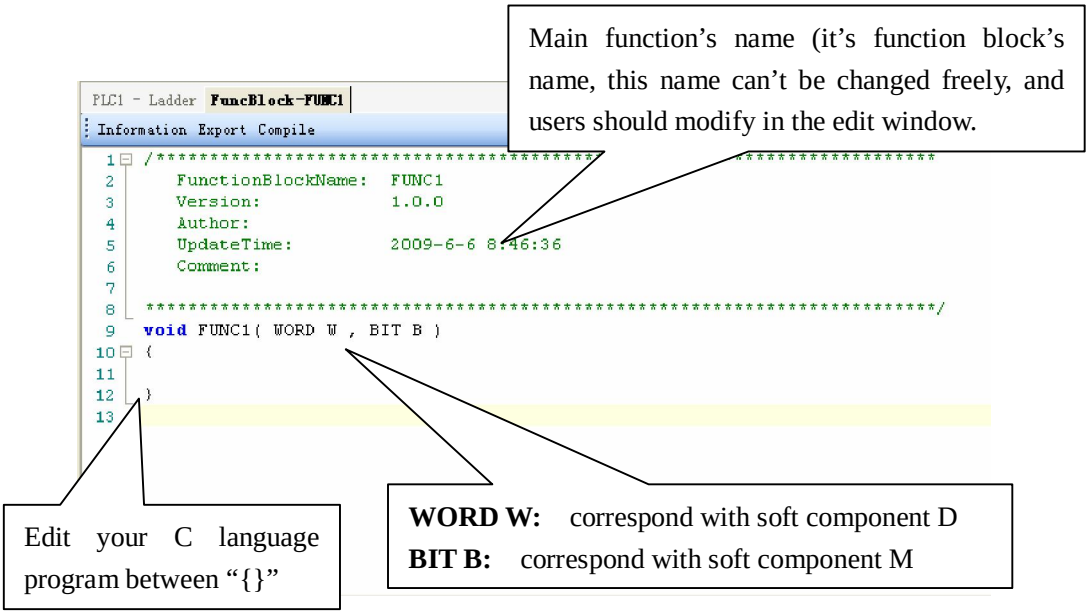
1、Open PLC edit tool, in the left "Project" toolbar, choose "Func Block", right click it and choose"Add New Func Block"



2、See graph below, fill in the information of your function;



3、After new create the Func Block, you can see the edit interface as shown below:



- Parameters' transfer format: if call the **Func Block** in ladder, the transferred D and M is the start ID of W and B. Take the above graph as the example, start with D0 and M0, then W[0] is D0, W[10] is D10, B［0 is M0, B［10］ is M10. If in the ladder the used parameters are D100, M100, then W[0] is D100, B［0］ is M100. So, word and bit component's start address is defined in PLC program by the user.
- Parameter W: represent **Word** soft component, use in the form of data group. E.g. W[0]=1;W[1]=W[2]+W[3]; in the program, use according to standard C language

rules.

- Parameter B: represent **Bit** soft component, use in the form of data group. Support **SET** and **RESET**. E.g: B[0]=1;B[1]=0; And assignment, for example B[0]=B[1]。
- Double-word operation: add **D** in front of **W**, e.g. DW[10]=100000, it means assignment to the double-word W[10]W[11]
- Floating Operation: Support the definition of floating variable in the function, and execute floating operation;
- Function Library: In **Func Block**, users can use the Functions and Variables in function library directly. For the Functions and Variables in function library, see the list in Appendix.
- The other data type supported:

```
BOOL;        //BOOL Quantity
INT8U;       //8 bits unsigned integral
INT8S;       //8 bits signed integral
INT16U       //16 bits unsigned integral
INT16S       //8 bits signed integral
INT32U       //32 bits unsigned integral
INT32S       //32 bits signed integral
FP32;        //Single precision Floating
FP64;        // Doubleprecision Floating
```

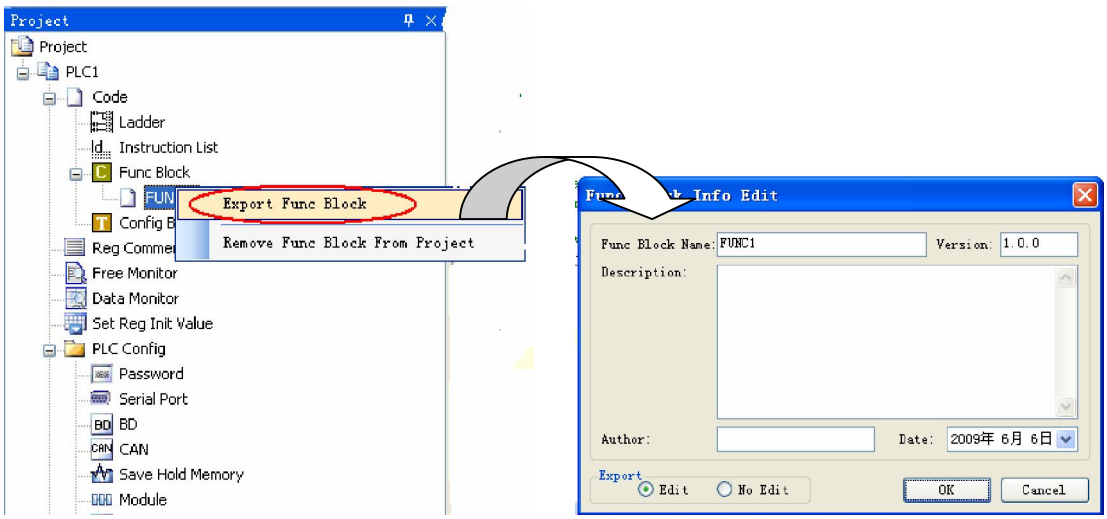
- Predefined Marco

```
#define    true     1
#define    false    0
#define    TRUE     1
#define    FALSE    0
```

9-4. Import and Export the Functions

1、Export

(1) Function: export the function as the file, then other PLC program can import to use;



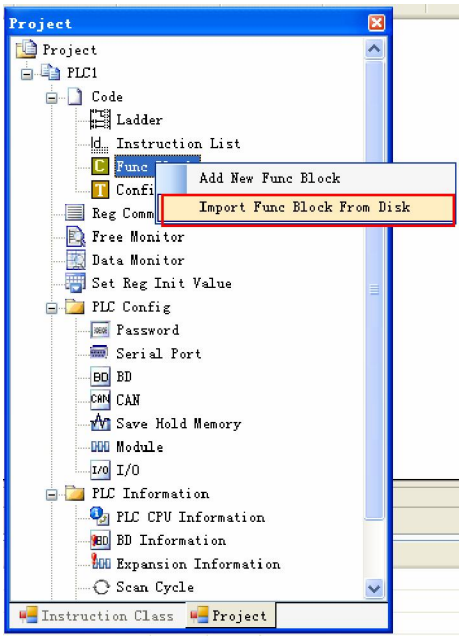
(2) Export Format

a) Editable; export the source codes out and save as a file. If import again, the file is editable;

b) Not editable: don't export the source code, if import the file, it's not editable;

2、Import

Function; Import the exist **Func Block** file, to use in the PLC program;



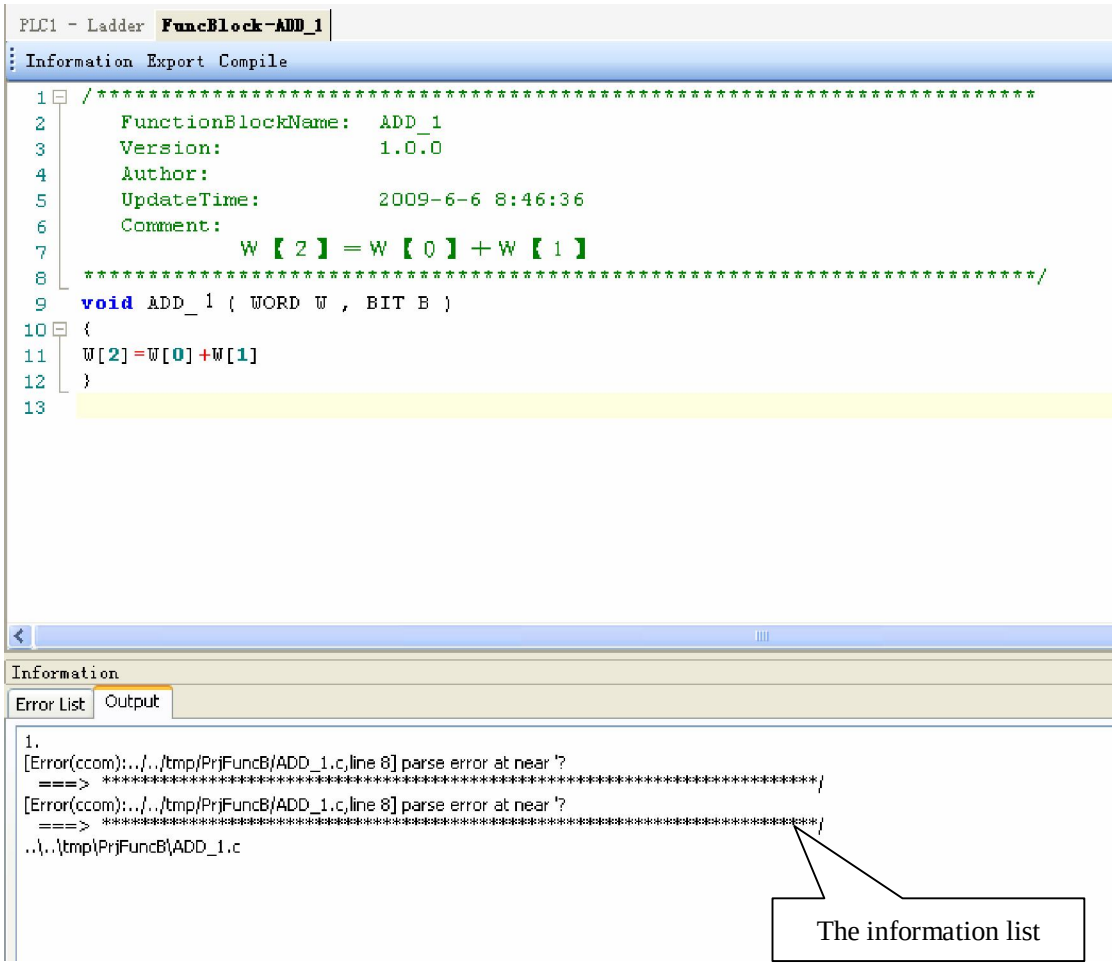
Choose the **Func Block**, right click "Import Func Block From Disk", choose the correct file, then click OK.

9-5. Edit eh Func Blocks

Example: Add D0 and D1 in PLC's registers, then assign the value to D2;

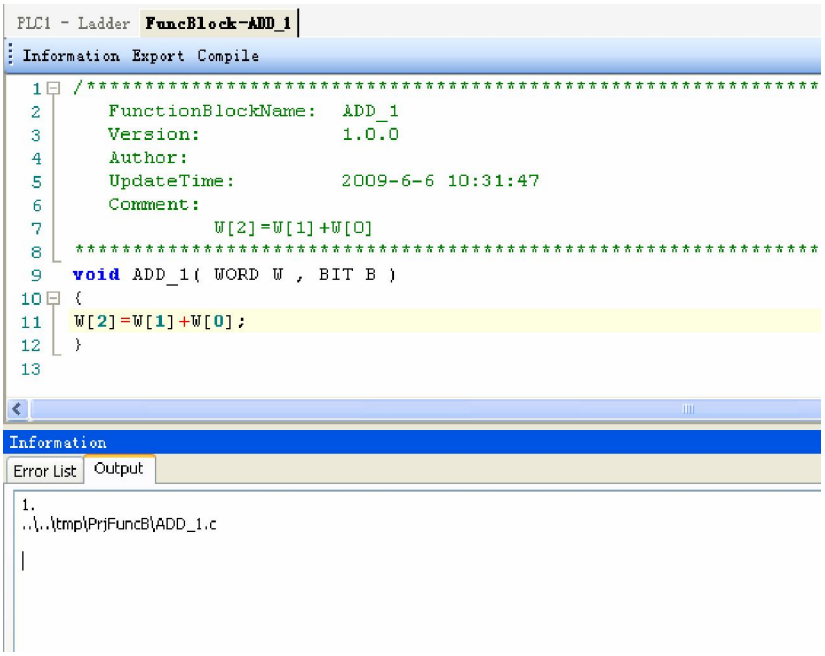
(1) In "Project" toolbar, new create a **Func Block**, here we name the **Func Block** as **ADD_2**, then edit C language program;

(2) Click **compile** after edition

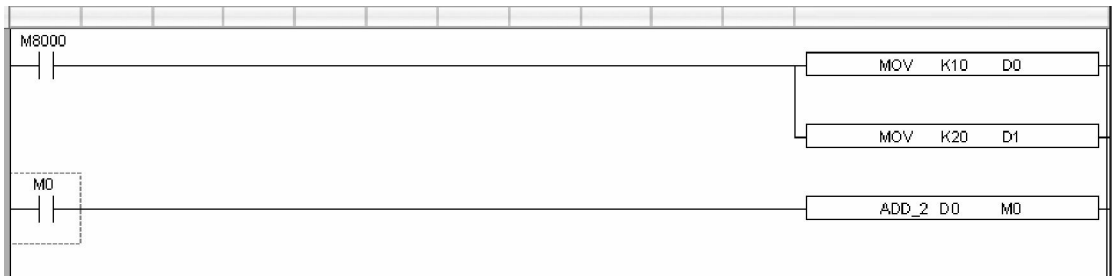


According to the information shown in the output blank, we can search and modify the grammar error in C language program. Here we can see that in the program there is no "**;**" sign behind W[2]=W[0]+W[1];

Compile the program again after modify the program. In the information list, we can corfirm that there is no grammar error in the program;

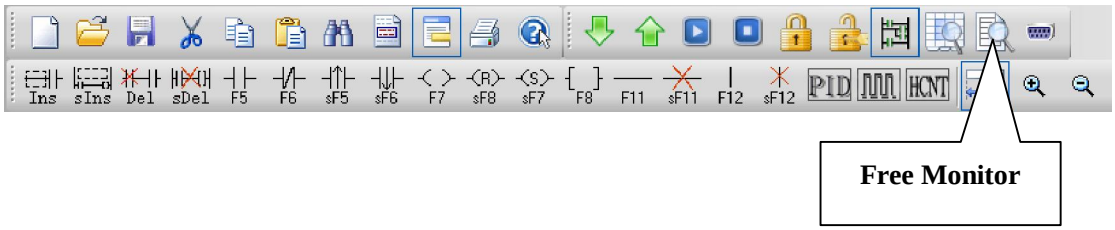


(3) Write PLC program, assign value 10 and 20 into registers D0, D1 separately, then call Func Block ADD_2, see graph below:



(4) Download program into PLC, run PLC and set M0.

(5) From Free Monitor in he toolbar, we can see that D2 changes to be 30, it means the assignment is successful;



9-6. Program Example

- Function: calculate CRC parity value via Func Block
- CRC calculation rules:

(1) Set 16 bits register (CRC register) = FFFF H

(2) XOR (Exclusive OR) 8 bits information with the low byte of the 16 bits CRC register.

(3) Right shift 1 bit of CRC register, fill 0 in the highest bit.

(4) Check the right shifted value, if it is 0, save the new value from step3 into CRC register; if it is not 0, XOR the CRC register value with A001 H and save the result into the CRC register.

(5) Repeat step3&4 until all the 8 bits have been calculated.

(6) Repeat step2~5, then calculate the next 8 bits information. Until all the information has been calculated, the result will be the CRC parity code in CRC register.

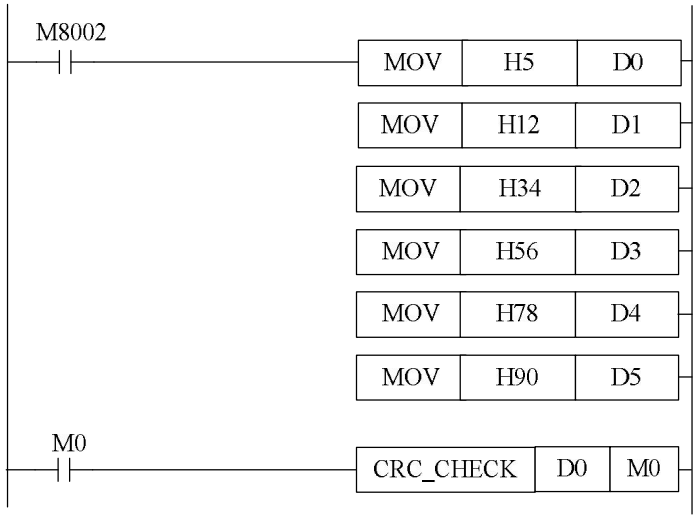
- Edit C language Func Block program, see graph below:

```
 9  void CRC_CHECK( WORD W , BIT B )
10  {
11      int i,j,m,n;
12      unsigned int reg_crc=0xffff,k;
13
14      for( i = 0 ; i < W[0] ; i++ )
15          {
16          reg_crc^=W[i+1];
17          for(j=0;j<8;j++)
18          {
19          if(reg_crc&0x01)
20              reg_crc=(reg_crc>>1)^0xa001;
21          else
22              reg_crc=reg_crc>>1;
23          }
24          }
25
26          m=W[0]+1;
27          n=W[0]+2;
28          k=reg_crc&0xff00;
29          W[m] = k>>8;
30          W[n]=reg_crc&0xff;
31         }
```

- Edit PLC ladder program,

D0: Parity data byte number;

D1~D5: Parity data's content, see graph below:



- Download to PLC, then RUN PLC, set M0, via Free Monitor, we can find that values in D6 and D7 are the highest and lowest bit of CRC parity value;

9-7. Application Points

- When upload the PLC program in which there are some Func Blocks, the Func Blocks can't be uploaded, there will be an error say: There is an unknown instruction;
- In one Func Block file, you can write many subsidry functions, can call each other;
- Each Func Block files are independent, they can't call its owned functions;
- Func Block files can call C language library functions in form of floating, arithmetic like sin, cos, tan etc.

9-8. Function Table

The default function library

Constant	Data	Description
_LOG2	(double)0.693147180559945309417232121458	Logarithm of 2
_LOG10	(double)2.302585092994045901093613792909 3	Logarithm of 10
_SQRT2	(double)1.414213562373095048801688724 21	Radical of 2
_PI	(double)3.141592653589793238462643383279 5	PI
_PIP2	(double)1.570796326794896619231321691639 75	PI/2
_PIP2x3	(double)4.712388980384689857693965074919 25	PI*3/2

String Function	Description
void * memchr(const void *s, int c, size_t n);	Return the first **c** position among **n** words before **s** position
int memcmp(const void *s1, const void *s2, size_t n);	Compare the first **n** words of position **s1** and **s2**
void * memcpy(void *s1, const void *s2, size_t n);	Copy **n** words from position **s2** to s1and return **s1**
void * memset(void *s, int c, size_t n);	Replace the **n** words start from **s** position with word **c**, and return position **s**
char * strcat(char *s1, const char *s2);	Connect string **ct** behind string **s**
char * strchr(const char *s, int c);	Return the first word **c** position in string **s**
int strcmp(const char *s1, const char *s2);	Compare string s1 and s2
char * strcpy(char *s1, const char *s2);	Copy string s1 to string s2

Double-precision math function	Single-precision math function	Description
double acos(double x);	float acosf(float x);	Inverse cosine function.
double asin(double x);	float asinf(float x);	Inverse sine function
double atan(double x);	float atanf(float x);	Inverse tangent function
double atan2(double y, double x);	float atan2f(float y, float x);	Inverse tangent value of parameter (y/x)
double ceil(double x);	float ceilf(float x);	Return the smallest double integral which is greater or equal with parameter **x**
double cos(double x);	float cosf(float x);	Cosine function
double cosh(double x);	float coshf(float x);	Hyperbolic cosine function cosh(x)=(e^x+e^(-x))/2.
double exp(double x);	float expf(float x);	Exponent (e^x) of a nature data
double fabs(double x);	float fabsf(float x);	Absolute value of parameter x

double floor(double x);	float floorf(float x);	Return the largets dounble integral which is smaller or equals with **x**
double fmod(double x, double y);	float fmodf(float x, float y);	If **y** is not zero, return the reminder of floating **x/y**
double frexp(double val, int _far *exp);	float frexpf(float val, int _far *exp);	Break floating data **x** to be mantissa and exponent **x** = m*2^exp, return the mantissa of m, save the logarithm into **exp**.
double ldexp(double x, int exp);	float ldexpf(float x, int exp);	X multipy the (two to the power of n) is x*2^n.
double log(double x);	float logf(float x);	Nature logarithm logx
double log10(double x);	float log10f(float x);	logarithm (log10x)
double modf(double val, double *pd);	float modff(float val, float *pd);	Break floating data X to be integral part and decimal part, return the decimal part, save the integral part into parameter ip.
double pow(double x, double y);	float powf(float x, float y);	Power value of parameter **y** (x^y)
double sin(double x);	float sinf(float x);	sine function
double sinh(double x);	float sinhf(float x);	Hyperbolic sine function, sinh(x)=(e^x-e^(-x))/2.
double sqrt(double x);	float sqrtf(float x);	Square root of parameter X
double tan(double x);	float tanf(float x);	tangent function.
double tanh(double x);	float tanhf(float x);	Hyperbolic tangent function, tanh(x)=(e^x-e^(-x))/(e^2+e^(-x)).