1. sizeof是C/C++中的一个操作符（operator）是也，简单的说其作用就是返回一个对象或者类型所占的内存字节数。

2. sizeof的三种语法形式

1) sizeof( object ); // sizeof( 对象 );

2) sizeof( type\_name ); // sizeof( 类型 );

3) sizeof object; // sizeof 对象;

3. sizeof的常量性：sizeof的计算发生在编译时刻，所以它可以被当作常量表达式使用，如：

char ary[sizeof(int)\*5];

sizeof会根据最终结果来计算大小，不会对表达式进行计算。

char stra[] = "HelloWorld";

char \*strb = stra;

cout <<"C " << sizeof(stra) << " D " << sizeof(strb++) << endl;

cout << strb << endl; // 输出为HelloWorld，sizeof会根据最终的结果来确定大小，但不会对表达式进行计算，因为其是在编译时刻计算的，只考虑对象的类型

sizeof(函数)：

返回函数返回值类型的大小，不会调用函数。

4. 指针变量的sizeof

指针记录了另一个对象的地址。既然是来存放地址的，那么它当然等于计算机内部地址总线的宽度。所以在32位计算机中，一个指针变量的返回值必定是4（注意结果是以字节为单位）。

char a[50];

int b[10];

char c[] = "abcde";

printf("%d\n", sizeof(char)); // 1

printf("%d\n", sizeof(int)); // 4

printf("%d\n", sizeof(float)); // 4

printf("%d\n", sizeof(double)); // 8

printf("%d\n", sizeof(long)); // 4

printf("%d\n", sizeof(short)); // 2

printf("%d\n", sizeof(a)); // 50 字符数组a的长度为50，每个字符占1个字节

printf("%d\n", sizeof(b)); // 40 字符数组b的长度为10，每个字符占4个字节

printf("%d\n", sizeof(c)); // 6，字符数组c的长度为6，后面还有一个字符串结束标志’\0’

需要注意下面的情况：

void func(int b[10]);

int main(int argc, char\* argv[])

{

int b[10];

printf("%d\n", sizeof(b)); // 40

return 0;

}

void func(int b[10])

{

printf("%d\n", sizeof(b)); // 输出为4，原因是b在C编译过程中被当成一个指针处理。

}

7. 结构体的sizeof

例：程序sizeof\_test1

struct A

{

char c;

int i;

};

int main(int argc, char\* argv[])

{

int a = 0;

a = sizeof(struct A);

printf("%d\n", a); // 输出为8

return 0;

}

char占一个字节，int占4个字节，但是结构体S1的长度为8个字节。这是因为字节对齐的原因。字节对齐有助于加快计算机的取数速度，否则就得多花指令周期了。为此，编译器默认会对结构体进行处理（实际上其它地方的数据变量也是如此），让宽度为2的基本数据类型（short等）都位于能被2整除的地址上，让宽度为4的基本数据类型（int等）都位于能被4整除的地址上，以此类推。这样，两个数中间就可能需要加入填充字节，所以整个结构体的sizeof值就增长了。

字节对齐的三原则：

1) 结构体变量的首地址能够被其最宽基本类型成员的大小所整除；

2) 结构体每个成员相对于结构体首地址的偏移量（offset）都是成员大小的整数倍（如果成员是复合成员，则其偏移量为复合成员的最宽基本类型来确定），如有需要编译器会在成员之间加上填充字节（internal adding）；

3) 结构体的总大小为结构体最宽基本类型成员大小的整数倍，如有需要编译器会在最末一个成员之后加上填充字节（trailing padding）。

基本类型是指前面提到的像char、short、int、float、double这样的内置数据类型，这里所说的“数据宽度”就是指其sizeof的大小。由于结构体的成员可以是复合类型，比如另外一个结构体，所以在寻找最宽基本类型成员时，应当包括复合类型成员的子成员，而不是把复合成员看成是一个整体。

结构体的大小等于最后一个成员的偏移量加上其大小再加上末尾的填充字节数目，即：

sizeof( struct ) = offsetof( last item ) + sizeof( last item ) + sizeof( trailing padding )

其中，trailing padding（末尾字节填充数）不一定有，如果offsetof( last item ) + sizeof( last item )为最宽基本类型成员大小的整数倍，trailing padding就为0，否则就需要填充字节。

看下例

例：程序sizeof\_test1

struct A

{

char c;

int i;

};

struct B

{

char c1;

struct A a1;

char c2;

};

struct C

{

int a;

char i;

};

int main(int argc, char\* argv[])

{

int a = 0;

int b = 0;

int c = 0;

a = sizeof(struct A);

b = sizeof(struct B);

c = sizeof(struct C);

printf("%d\n", a); // 8

printf("%d\n", b); // 16

printf("%d\n", c); // 8

return 0;

}

根据以上的字节对齐原则，对于结构体A，字符c的偏移量为0（本身就是首地址），整型变量i的偏移量为4（因为整型变量的大小为4），整型变量本身的大小为4，相加后结构体A的大小为8。其内存分布如下：

字符c 填充字节 填充字节 填充字节 整型变量i 整型变量i 整型变量i 整型变量i

字符c后面有3个填充字节，之后的4个字节存储整型变量i。

结构体C的大小也为8，这是因为，字符i的偏移量为4（因为前面存储了整型变量a），本身大小为1，相加后为5，根据字节对齐原则（3），末尾需补充3个字节，其内存分布如下：

整型变量a 整型变量a 整型变量a 整型变量a 字符i 填充字节 填充字节 填充字节

前4个字节存储整型变量a，第5个字节存储字符i，后面跟3个填充字节

结构体b1的大小为16，这是由于：A的最宽基本成员的类型为int，B在考虑最宽基本类型成员时是将A“打散”看的，所以B的最宽基本类型为int，这样，通过B定义的变量，其存储空间首地址需要被4整除，整个sizeof(B)的值也应该被4整除。c1的偏移量为0，a1的偏移量是将a1看成一个整体，它作为结构体变量也满足前面三个准则，所以其大小为8，偏移量为4（这是根据a1中int a来确定的），c1与a1之间便需要3个填充字节，而c2与a1之间就不需要了，所以c2的偏移量为12，算上c2的大小为13，13是不能被4整除的，这样末尾还得补上3个填充字节。最后得到sizeof(B)的值为16。

结构B

字符c1 填充字节 填充字节 填充字节 变量a1 变量a1 变量a1 变量a1 变量a1 变量a1 变量a1 变量a1 字符c2 填充字节 填充字节 填充字节

看下例：

例：程序sizeof\_test2

struct A

{

char i;

double a;

};

struct C

{

double a;

char i;

};

struct B

{

char c1;

struct C c2;

char c3;

};

int main(int argc, char\* argv[])

{

int a = 0;

int b = 0;

int c = 0;

a = sizeof(struct A);

b = sizeof(struct B);

c = sizeof(struct C);

printf("%d\n", a); // 16

printf("%d\n", b); // 32

printf("%d\n", c); // 16

return 0;

}

C的内存为：cc（填充字节）

a a a a a a a a i cc cc cc cc cc cc cc

A的内存为：

i cc cc cc cc cc cc cc a a a a a a a a

结构体b1的大小为32，c1的偏移量为0，结构体c2的偏移量为8（按c的double 型成员a来算），c3的偏移量为24，本身大小为1，相加为25，不是8的倍数，所以要填充字节。B的大小就为32

B的内存：

c1 cc cc cc cc cc cc cc c2 c2 c2 c2 c2 c2 c2 c2 c2 c2 c2 c2 c2 c2 c2 c2 c3 cc cc cc cc cc cc cc

对于结构体中有数组的情况

例：程序sizeof\_test3

struct A

{

double a;

char i[11];

int k;

};

struct C

{

char i[11];

double a;

int k;

};

struct B

{

char c1;

struct A a1;

};

union S // 联合体

{

char c1;

struct B b1;

};

int main(int argc, char\* argv[])

{

int a = 0;

int b = 0;

int c = 0;

int s = 0;

a = sizeof(struct A);

b = sizeof(struct B);

c = sizeof(struct C);

s = sizeof(union S);

printf("%d\n", a); // 24

printf("%d\n", b); // 32

printf("%d\n", c); // 32

printf("%d\n", s); // 32

return 0;

}

结构体A的内存分布为：填充字节cc

a a a a a a a a i i i i i i i i i i i cc k k k k

结构体C的内存分布为：填充字节cc

i i i i i i i i i i i cc cc cc cc cc a a a a a a a a k k k k cc cc cc cc

结构体B的大小为32，其中c1的偏移量为0，而a1的偏移量为8（是按double型变量，而不是字符数组）

对于联合体S，其sizeof的值就是其各个成员变量sizeof的最大值，所以为32.

union S1

{

char c1;

int a[10];

double b;

};

其字节大小为40，最大的成员变量为数组a，字节大小为40.