NAME

sizeof operator - yield the storage size of the given operand

SYNTAX

sizeof (*type*)
sizeof *expression*

DESCRIPTION

The **sizeof** operator yields the size of its operand. The **sizeof** operator cannot be applied to incomplete types and expressions with incomplete types (e.g. *void*, or forward-defined *struct foo)*, and function types.

The size of primitive (non-derived) data types in C may differ across hardware platforms and implementations. They are defined by corresponding Application Binary Interface (ABI) specifications, see arch(7) for details about ABI used by FreeBSD. It may be necessary or useful for a program to be able to determine the storage size of a data type or object to account for the platform specifics.

The unary **sizeof** operator yields the storage size of an expression or data type in *char sized units* (C language bytes). As a result, ‘sizeof(char)’ is always guaranteed to be 1. (The number of bits per *char* is given by the CHAR_BIT definition in the <*limits.h*> header; many systems also provide the "number of bits per byte" definition as NBBY in the <*sys/param.h*> header.)

EXAMPLES

Different platforms may use different data models. For example, systems on which integers, longs, and pointers are using 32 bits (e.g., i386) are referred to as using the "ILP32" data model, systems using 64 bit longs and pointers (e.g., amd64 / x86_64) as the "LP64" data model.

The following examples illustrate the possible results of calling **sizeof** on an ILP32 vs. an LP64 system:

When applied to a simple variable or data type, **sizeof** returns the storage size of the data type of the object:

Object or type	**Result (ILP32)**	**Result (LP64)**
sizeof(char)	1	1
sizeof(int)	4	4
sizeof(long)	4	8
sizeof(float)	4	4
sizeof(double)	8	8
sizeof(char *)	4	8

For initialized data or uninitialized arrays of a fixed size known at compile time, **sizeof** will return the correct storage size:

```
#define DATA "1234567890"
char buf1[] = "abc";
char buf2[1024];
char buf3[1024] = { ’a’, ’b’, ’c’ };
```

Object or type	**Result**
sizeof(DATA)	11
sizeof(buf1)	4
sizeof(buf2)	1024
sizeof(buf3)	1024

The examples above are the same for ILP32 and LP64 platforms, as they are based on character units.

When applied to a struct or union, **sizeof** returns the total number of bytes in the object, including any internal or trailing padding used to align the object in memory. This result may thus be larger than if the storage size of each individual member had been added:

```
struct s1 {
        char c;
};

struct s2 {
        char *s;
        int i;
};

struct s3 {
        char *s;
        int i;
        int j;
};

struct s4 {
                int i;
                uint64_t i64;
};
```

```
struct s5 {
        struct s1 a;
        struct s2 b;
        struct s3 c;
        struct s4 d;
};
```

Object or type	Result (ILP32)	Result (LP64)
sizeof(struct s1)	1	1
sizeof(struct s2)	8	16
sizeof(struct s3)	12	16
sizeof(struct s4)	12	16
sizeof(struct s5)	36	56

When applied to a struct containing a flexible array member, **sizeof** returns the size of the struct *without* the array, although again possibly including any padding the compiler deemed appropriate:

```
struct flex {
        char c;
        long b;
        char array[];
}
```

Object or type	Result (ILP32)	Result (LP64)
sizeof(struct flex)	8	16

One of the more common uses of the **sizeof** operator is to determine the correct amount of memory to allocate:

```
int *nums = calloc(512, sizeof(int));
```

The **sizeof** operator can be used to calculate the number of elements in an array by dividing the size of the array by the size of one of its elements:

```
int nums[] = { 1, 2, 3, 4, 5 };
const int howmany = sizeof(nums) / sizeof(nums[0]);
```

Many systems provide this shortcut as the macro ntimes() via the <*sys/param.h*> header file.

RESULT

The result of the **sizeof** operator is an unsigned integer type, defined in the stddef.h header as a *size_t*.

NOTES

It is a common mistake to apply **sizeof** to a dynamically allocated array:

```
char *buf;
if ((buf = malloc(BUFSIZ)) == NULL) {
        perror("malloc");
}
/* Warning: wrong! */
(void)strncat(buf, input, sizeof(buf) - 1);
```

In that case, the operator will return the storage size of the pointer (‘sizeof(char *)’), not the allocated memory.

sizeof determines the size of the result of the expression given, but *does not* evaluate the expression:

```
int a = 42;
printf("%ld - %d\n", sizeof(a = 10), a); /* Result: "4 - 42" */
```

Since it is evaluated by the compiler and not the preprocessor, the **sizeof** operator cannot be used in a preprocessor expression.

SEE ALSO

arch(7), operator(7)

STANDARDS

The **sizeof** operator conforms to ANSI X3.159-1989 ("ANSI C89").

Handling of flexible array members in structures conforms to ISO/IEC 9899:1999 ("ISO C99").

AUTHORS

This manual page was written by Jan Schaumann <*jschauma@netmeister.org*>.