# Appendix-D: *Operator Keywords*

The following are the statement keywords in C++. Only the highlighted will be describe in detail.

| | |
|---|---|
| **delete** | **typeid** |
| **operator** | **new** |
| **sizeof** | |

## delete

**Syntax**
<::> delete <cast-expression>
<::> delete [ ] <cast-expression>
delete <array-name> [ ];

**Description**
The delete operator offer dynamic storage deallocation, deallocating a memory block allocated by a previous call to new. It is similar but superior to the standard library function free.
You can also use the delete operator to remove arrays that you no longer need.

## operator

**Syntax**
operator <operator symbol>( <parameters> )
{
  <statements>;
}

**Description**
Use the operator keyword to define a new (overloaded) action of the given operator. When the operator is overloaded as a member function, only one argument is allowed, as *this is implicitly the first argument. When you overload an operator as a friend, you can specify two arguments.

## sizeof

**Syntax**
sizeof <expression>
sizeof ( <type> )

**Description**
Use the sizeof operator to return the size, in bytes, of the given expression or type (as type size_t ).

## typeid

**Syntax**
typeid( expression )
typeid( type-name )

**Description**
You can use typeid to get run-time identification of types and expressions. A call to typeid returns a reference to an object of type const typeinfo. The returned object represents the type of the typeid operand. If the typeid operand is a dereferenced pointer or a reference to a polymorphic type, typeid returns the dynamic type of the actual object pointed or referred to. If the operand is non-polymorphic, typeid returns an object that represents the static type.
You can use the typeid operator with fundamental data types as well as user-defined types.
If the typeid operand is a dereferenced NULL pointer, the Bad_typeid exception is thrown.

# new

**Syntax**

<::> new <new-args> type-name <(initializer)>
<::> new <new-args> (type-name) <(initializer)>

**Description**

The new operator offer dynamic storage allocation, similar but superior to the standard library function malloc. The new operator must always be supplied with a data type in place of type-name. Items surrounded by angle brackets are optional. The optional arguments can be as follows:

:: operator, invokes the global version of new.

new-args can be used to supply additional arguments to new. You can use this syntax only if you have have an overloaded version of new that matches the optional arguments.

initializer, if present is used to initialize the allocation.

A request for non-array allocation uses the appropriate operator new() function. Any request for array allocation will call the appropriate operator new[]() function. Selection of operator is done as follows:

By default, operator new[]() calls operator new()

If a class Type has an overloaded version of operator new[](), arrays of Type will be allocated using Type::operator new[]()

If a class Type has an overloaded version of new, and no overloaded version of the array allocator operator new[](), arrays of Type will be allocated using Type::operator new()

Memory for a non-array object of Type is allocated using Type::operator new()

If none of the above cases apply, the global ::operator new() is used

**Note:** Arrays of classes require the default constructor.

new tries to create an object of type Type by allocating (if possible) sizeof(Type) bytes in free store (also called the heap). new calculates the size of Type without the need for an explicit sizeof operator. Further, the pointer returned is of the correct type, "pointer to Type," without the need for explicit casting. The storage duration of the new object is from the point of creation until the operator delete destroys it by deallocating its memory, or until the end of the program.
If successful, new returns a pointer to the new object. By default, an allocation failure (such as insufficient or fragmented heap memory) results in the predefined exception xalloc being thrown. Your program should always be prepared to catch the xalloc exception before trying to access the new object (unless you use a new-handler).

A request for allocation of 0 bytes returns a non-null pointer. Repeated requests for zero-size allocations return distinct, non-null pointers.

Handling Errors for the new Operator

You can define a function to be called if the new operator fails. To tell the new operator about the new-handler function, use set_new_handler and supply a pointer to the new-handler. If you want new to return null on failure, you must use set_new_handler(0).