# Overloading New and Delete operator in c

*GeeksforGeeks*

The new and delete operators can also be overloaded like other operators in C++. New and Delete operators can be overloaded globally or they can be overloaded for specific classes.

If these operators are overloaded using member function for a class, it means that these operators are overloaded **only for that specific class**.

If overloading is done outside a class (i.e. it is not a member function of a class), the overloaded 'new' and 'delete' will be called anytime you make use of these operators (within classes or outside classes). This is **global overloading**.

**The syntax for overloading the new operator :**

**void* operator new(size_t size);**

The overloaded new operator receives size of type size_t, which specifies the number of bytes of memory to be allocated. The return type of the overloaded new must be void*.The overloaded function returns a pointer to the beginning of the block of memory allocated.
**Syntax for overloading the delete operator :**

**void operator delete(void*);**

The function receives a parameter of type void* which has to be deleted. Function should not return anything.
**NOTE:** Both overloaded new and delete operator functions are **static members** by default. Therefore, they don't have access to **this pointer** .
**Overloading new and delete operator for a specific class:**

CPP

## CPP

```
#include<iostream>

#include<stdlib.h>

using namespace std;

class student

{

    string name;
    int age;
public:
    student()
    {
        cout<< "Constructor is called\n" ;
    }
    student(string name, int age)
    {
        this->name = name;
        this->age = age;
    }
    void display()
    {
        cout<< "Name:" << name << endl;
        cout<< "Age:" << age << endl;
    }
    void * operator new(size_t size)
    {
        cout<< "Overloading new operator with size: " << size << endl;
        void * p = ::operator new(size);
        return p;
    }
    void operator delete(void * p)
    {
        cout<< "Overloading delete operator " << endl;
        free(p);
    }
};
```

```
int main()

{

    student * p = new student("Yash", 24);

    p->display();

    delete p;

}
```

**Output**

```
Overloading new operator with size: 40
Name:Yash
Age:24
Overloading delete operator
```

**NOTE:** In the above new overloaded function, we have allocated dynamic memory through new operator, but it should be global new operator otherwise it will go in recursion
void *p = new student(); // this will go in recursion as**new** will be overloaded again and again
void *p = ::new student(); // this is correct

**Global overloading of new and delete operator**

CPP

## CPP

```
#include<iostream>

#include<stdlib.h>

using namespace std;

void * operator new(size_t size)

{

    cout << "New operator overloading " << endl;

    void * p = malloc(size);

    return p;

}

void operator delete(void * p)

{

    cout << "Delete operator overloading " << endl;
    free(p);

}

int main()

{

    int n = 5, i;

    int * p = new int[n];

    for (i = 0; i<n; i++)

    p[i]= i;

    cout << "Array: ";

    for(i = 0; i<n; i++)

        cout << p[i] << " ";

    cout << endl;

    delete [] p;

}
```

**Output**

```
New operator overloading
Array: 0 1 2 3 4
Delete operator overloading
```

**NOTE:** In the code above, in new overloaded function we cannot allocate memory using **::new int[5]** as it will go in recursion. We need to allocate memory using malloc only.

**Why to overload new and delete?**

1. The overloaded new operator function can accept arguments; therefore, a class can have **multiple overloaded new operator** functions. This gives the programmer more flexibility in customizing memory allocation for objects. For example:

C

## C

```
void *operator new(size_t size, char c)

{

   void *ptr;
   ptr = malloc(size);

   if (ptr!=NULL)

      *ptr = c;

return ptr;

}

main()

{

   char *ch = new('#') char;

}
```

**2. NOTE:** Code will not only allocate memory for single character but will also initialize the allocated memory with **# character**.

3. Overloaded new or delete operators also provide **Garbage Collection** for class's object.

**4. Exception handling routine** can be added in overloaded new operator function.

5. Sometimes you want operators new and delete to do something **customized that the compiler-provided versions don't offer**. For example, You might write a custom operator delete that overwrites deallocated memory with zeros in order to increase the security of application data.

6. We can use **realloc() function** in new function to re-allocate memory dynamically.

7. Overloaded new operator also enables programmers to squeeze some **extra performance** out of their programs. For example, In a class, to speed up the allocation of new nodes, a list of deleted nodes is maintained so that their memory can be reused when new nodes are allocated.In this case, the overloaded delete operator will add nodes to the list of deleted nodes and the overloaded new operator will allocate memory from this list rather than from the heap to speedup memory allocation. Memory from the heap can be used when the list of deleted nodes is empty.