# C++ Memory Management: new and delete



Arrays can be used to store multiple homogenous data but there are serious drawbacks of using arrays.

You should allocate the memory of an array when you declare it but most of the time, the exact memory needed cannot be determined until runtime.

The best thing to do in this situation is to declare an array with maximum possible memory required (declare array with maximum possible size expected).

The downside to this is unused memory is wasted and cannot be used by any other programs.

To avoid wastage of memory, you can dynamically allocate memory required during runtime using new and delete operator in C++.

## Example 1: C++ Memory Management

**C++ Program to store GPA of**n**number of students and display it where**n**is the number of students entered by user.**

#include <iostream>

#include <cstring>

using namespace std;

int main()

{

int num;

cout << "Enter total number of students: ";

cin >> num;

float\* ptr;

// memory allocation of num number of floats

ptr = new float[num];

cout << "Enter GPA of students." << endl;

for (int i = 0; i < num; ++i)

{

cout << "Student" << i + 1 << ": ";

cin >> \*(ptr + i);

}

cout << "\nDisplaying GPA of students." << endl;

for (int i = 0; i < num; ++i) {

cout << "Student" << i + 1 << " :" << \*(ptr + i) << endl;

}

// ptr memory is released

delete [] ptr;

return 0;

}

**Output**

Enter total number of students: 4

Enter GPA of students.

Student1: 3.6

Student2: 3.1

Student3: 3.9

Student4: 2.9

Displaying GPA of students.

Student1 :3.6

Student2 :3.1

Student3 :3.9

Student4 :2.9

In this program, only the memory required to store num (entered by user) number of floating-point data is declared dynamically.

## The new Operator

ptr = new float[num];

This expression in the above program returns a [pointer](https://www.programiz.com/cpp-programming/pointers) to a section of memory just large enough to hold the num number of floating-point data.

## The delete Operator

Once the memory is allocated using new operator, it should released back to the operating system.

If the program uses a large amount of memory using new, system may crash because there will be no memory available for the operating system.

The following expression returns memory back to the operating system.

delete [] ptr;

The brackets **[]** indicates the array has been deleted. If you need to delete a single object then, you don't need to use brackets.

delete ptr;

## Example 2: C++ Memory Management

**Object-oriented approach to handle above program in C++.**

#include <iostream>

using namespace std;

class Test

{

private:

int num;

float \*ptr;

public:

Test()

{

cout << "Enter total number of students: ";

cin >> num;

ptr = new float[num];

cout << "Enter GPA of students." << endl;

for (int i = 0; i < num; ++i)

{

cout << "Student" << i + 1 << ": ";

cin >> \*(ptr + i);

}

}

~Test() {

delete[] ptr;

}

void Display() {

cout << "\nDisplaying GPA of students." << endl;

for (int i = 0; i < num; ++i) {

cout << "Student" << i+1 << " :" << \*(ptr + i) << endl;

}

}

};

int main() {

Test s;

s.Display();

return 0;

}

The output of this program is same as the above program.

When the object s is created, the constructor is called which allocates the memory for numfloating-point data.