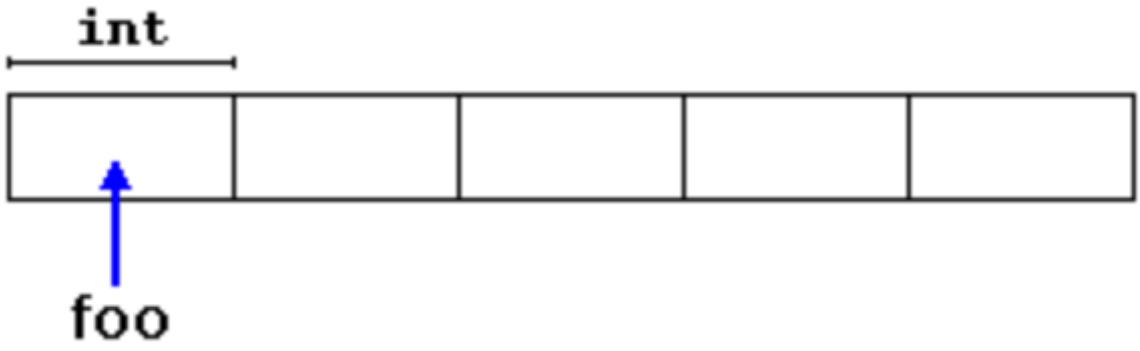
DYNAMIC MEMORY ALLOCATION (Dinamik Bellek Paylaştırma, Ayırma)
Dynamic memory is allocated using operator **new** and it is followed by a **data type specifier** and, if a sequence of more than one element is required, the number of these within brackets **[]**.
It returns a pointer to the beginning of the new block of memory allocated. Its syntax is:

```
pointer = new type
pointer = new type [number_of_elements]
```

The first expression is used to allocate memory to contain one single element of **type.**
The second one is used to allocate a block(an array) of elements of **type,** where **number_of_elements** is an integer value representing the amount of these.

int* foo;
foo = **new int [** 5 **]**;

The system dynamically allocates space for five elements of **type int** and returns a pointer to the first element of the sequence, which is assigned to foo (a pointer). Therefore, foo now points to a valid block of memory with space for five elements of type **int**.



Here, foo is a pointer, and thus, the first element pointed to by foo can be accessed either with the expression foo[0] or the expression *foo (both are equivalent).
The second element can be accessed either with foo[1] or *(foo+1), and so on...

The most important difference is that the size of a regular array needs to be a ***constant expression***, and thus its size has to be determined **at the moment of designing the program**, before it is run, whereas the dynamic memory allocation performed by **new** allows to assign memory **during runtime using any variable value as size.**

There are no guarantees that all requests to allocate memory using operator **new**

are going to be granted(izin verildi) by the system because of the limited resource of the computer.
C++ provides two standard mechanisms to check if the allocation was successful:
1)Exception of type = **bad_alloc** (the program execution is terminated.)

```
foo = new int [5];  // if allocation fails, an exception is thrown
```

2)**no throw** (the pointer returned by new is a *null pointer*, and the program continues its execution normally.)

```
foo = new (nothrow) int [5];
```

If the allocation of this block of memory fails, the failure **can be detected by checking if foo is a null pointer:**

```
1 int * foo;
2 foo = new (nothrow) int [5];
3 if (foo == nullptr) {
4   // error assigning memory. Take measures.
5 }
```

This nothrow method is likely to(muhtemelen) produce less efficient code than exceptions, since it implies explicitly checking the pointer value returned after each and every allocation. Therefore, the exception mechanism is generally preferred, at least for critical allocations. Still, most of the coming examples will use the nothrow mechanism due to its simplicity.

Operators delete and delete[ ]
In most cases, memory allocated dynamically is only needed during specific periods of time within a program; once it is no longer needed, it can be freed so that the memory becomes available again for other requests of dynamic memory. This is the purpose of operator delete, whose syntax is:

```
1 delete pointer;
2 delete[] pointer;
```

The first statement releases the memory of a single element allocated using new. The second one releases the memory allocated for arrays of elements using new and a size in brackets ([ ]).

```cpp
// rememb-o-matic
#include <iostream>
#include <new>
using namespace std;

int main ()
{
  int i,n;
  int * p;
  cout << "How many numbers would you like to type? ";
  cin >> i;
  p= new (nothrow) int[i];
  if (p == nullptr)
    cout << "Error: memory could not be allocated";
  else
  {
    for (n=0; n<i; n++)
    {
      cout << "Enter number: ";
      cin >> p[n];
    }
    cout << "You have entered: ";
    for (n=0; n<i; n++)
      cout << p[n] << ", ";
    delete[] p;
  }
  return 0;
}
```

```
How many numbers would you like to type? 5
Enter number : 75
Enter number : 436
Enter number : 1067
Enter number : 8
Enter number : 32
You have entered: 75, 436, 1067, 8, 32,
```

Dynamic Memory In C

C++ integrates the operators new and delete for allocating dynamic memory. But these were not available in the C language; instead, it used a library solution, with the functions malloc, calloc, realloc and free, defined in the header <cstdlib> (known as <stdlib.h> in C). The functions are also available in C++ and can also be used to allocate and deallocate dynamic memory.