Dynamic memory

In the programs seen in previous chapters, all memory needs were determined before program execution by defining the variables needed. But there may be cases where the memory needs of a program can only be determined during runtime. For example, when the memory needed depends on user input. On these cases, programs need to dynamically allocate memory, for which the C++ language integrates the operators `new` and `delete`.

在前几章看到的程序中，所有的内存需求都是在程序执行之前通过定义所需的变量来确定的。但是在某些情况下，程序的内存需求只能在运行时确定。例如，当所需的内存取决于用户输入时。在这些情况下，程序需要动态分配内存，为此c++语言集成了操作符new和delete。

Operators new and new[]

Dynamic memory is allocated using operator `new`. `new` 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:

动态内存使用operator new分配。New后面跟着一个数据类型说明符，如果需要一个包含多个元素的序列，则在括号[]中包含这些元素的数量。它返回一个指向新分配内存块开始位置的指针。它的语法是:

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

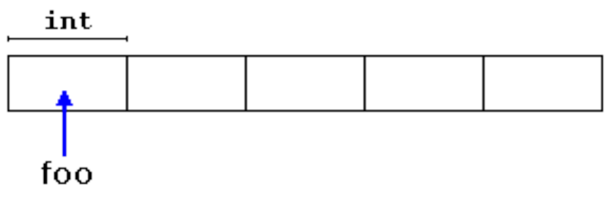
The first expression is used to allocate memory to contain one single element of type `type`. The second one is used to allocate a block (an array) of elements of type `type`, where `number_of_elements` is an integer value representing the amount of these. For example:

第一个表达式用于分配内存以包含type的单个元素。第二个用于分配一个类型为type的元素块(数组)，其中number_of_elements是一个表示这些元素数量的整数值。例如:

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

In this case, 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`.

在这种情况下，系统动态地为5个int类型的元素分配空间，并返回一个指向序列第一个元素的指针，该指针被赋给foo(一个指针)。因此，foo现在指向一个有效的内存块，其中有5个int类型的元素。



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...

在这里，foo是一个指针，因此，foo指向的第一个元素可以通过表达式foo[0]或表达式*foo访问(两者是等效的)。第二个元素可以通过foo[1]或*(foo+1)访问，等等...

There is a substantial difference between declaring a normal array and allocating dynamic memory for a block of memory using `new`. 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.

声明普通数组和使用new为内存块分配动态内存之间有很大的区别。最重要的区别是,一个常规数组中需要一个常数表达式,因此它的大小必须确定目前的设计程序,运行之前,而执行的动态内存分配新的允许在运行时分配内存使用任何变量值的大小。

The dynamic memory requested by our program is allocated by the system from the memory heap. However, computer memory is a limited resource, and it can be exhausted. Therefore, there are no guarantees that all requests to allocate memory using operator `new` are going to be granted by the system.

程序所请求的动态内存是由系统从内存堆中分配的。然而，计算机内存是一种有限的资源，它可能会被耗尽。因此，不能保证所有使用operator new分配内存的请求都将由系统授予。

C++ provides two standard mechanisms to check if the allocation was successful:

c++提供了两种标准机制来检查分配是否成功:

One is by handling exceptions. Using this method, an exception of type `bad_alloc` is thrown when the allocation fails. Exceptions are a powerful C++ feature explained later in these tutorials. But for now, you should know that if this exception is thrown and it is not handled by a specific handler, the program execution is terminated.

一是通过处理异常。使用此方法，当分配失败时抛出bad_alloc类型的异常。异常是一个强大的c++特性，在后面的教程中会解释。但是现在，您应该知道，如果抛出了这个异常，并且没有由特定的处理程序处理它，则程序执行将终止。

This exception method is the method used by default by `new`, and is the one used in a declaration like:

这个异常方法是new默认使用的方法，并且是在如下声明中使用的方法:

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

The other method is known as `nothrow`, and what happens when it is used is that when a memory allocation fails, instead of throwing a `bad_alloc` exception or terminating the program, the pointer returned by `new` is a *null pointer*, and the program continues its execution normally.

另一种方法是nothrow，当内存分配失败时，new返回的指针是一个空指针，程序继续正常执行，而不是抛出一个bad_alloc异常或终止程序。

This method can be specified by using a special object called `nothrow`, declared in header `<new>`, as argument for `new`:

这个方法可以通过使用一个名为nothrow的特殊对象来指定，该对象在头文件中声明，作为new的参数

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

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

在这种情况下，如果这个内存块的分配失败，失败可以通过检查foo是否是一个空指针来检测:

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

This `nothrow` method is likely to 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.

这个nothrow方法产生的代码可能比异常更低效，因为它意味着显式地检查每次分配后返回的指针值。因此，通常首选异常机制，至少对于关键的分配是这样的。不过，由于nothrow机制的简单性，接下来的大多数示例都将使用它。

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:

在大多数情况下，动态分配的内存只需要在程序中的特定时间段内使用;一旦不再需要它，就可以释放它，这样内存就可以再次用于其他动态内存请求。这是操作符' delete '的目的，其语法如下:

```
delete pointer;
delete[] pointer;
```

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

第一个语句释放使用new分配的单个元素的内存，第二个语句释放使用new和括号内的size([])分配的元素数组的内存。

The value passed as argument to `delete` shall be either a pointer to a memory block previously allocated with `new`, or a *null pointer* (in the case of a *null pointer*, `delete` produces no effect).

作为参数传递给delete的值必须是一个指针，该指针指向先前用new分配的内存块，或者是一个空指针(对于空指针，delete不起作用)。

```
// 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;
}
```

Notice how the value within brackets in the new statement is a variable value entered by the user (`i`), not a constant expression:

注意，new语句中括号内的值是用户输入的变量值，而不是常量表达式:

```
p= new (nothrow) int[i];
```

There always exists the possibility that the user introduces a value for `i` so big that the system cannot allocate enough memory for it. For example, when I tried to give a value of 1 billion to the "How many numbers" question, my system could not allocate that much memory for the program, and I got the text message we prepared for this case (`Error: memory could not be allocated`).

始终存在这样的可能性:用户为i引入的值太大，以至于系统无法为它分配足够的内存。例如，当我试图给“多少个数字”的问题一个10亿的值时，我的系统无法为程序分配那么多的内存，我得到了我们为这种情况准备的短信(Error: memory could not be assigned)。

It is considered good practice for programs to always be able to handle failures to allocate memory, either by checking the pointer value (if `nothrow`) or by catching the proper exception.

对于程序来说，总是能够处理分配内存的失败被认为是一种良好的实践，可以通过检查指针值(如果没有抛出)或通过捕获适当的异常。

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.

c++集成了operator new和delete来分配动态内存。但是这些在C语言中是不可用的;相反，它使用了一个标准库解决方案，函数malloc, calloc, realloc和free定义在头文件 (C中称为<stdlib.h>)。这些函数在c++中也可用，也可以用来分配和释放动态内存。

Note, though, that the memory blocks allocated by these functions are not necessarily compatible with those returned by `new`, so they should not be mixed; each one should be handled with its own set of functions or operators.

但是请注意，由这些函数分配的内存块不一定与new返回的内存块兼容，因此它们不应混合使用;每一个都应该用自己的一组函数或操作符来处理。