Compound Data Types:

12. Dynamic Memory

Until now, in all our programs, we have only had as much memory available as we declared for our variables, having the size of all of them to be determined in the source code, before the execution of the program. But, what if we need a variable amount of memory that can only be determined during runtime? For example, in the case that we need some user input to determine the necessary amount of memory space.

The answer is *dynamic memory*, for which C++ integrates the operators `new` and `delete`.

Operators new and new[]

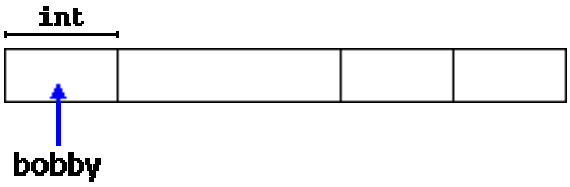
In order to request dynamic memory we use the operator `new`. `new` is followed by a data type specifier and -if a sequence of more than one element is required- the number of these within brackets `[]`. It returns a pointer to the beginning of the new block of memory allocated. Its form is:

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

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

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

In this case, the system dynamically assigns space for five elements of type `int` and returns a pointer to the first element of the sequence, which is assigned to `bobby`. Therefore, now, `bobby` points to a valid block of memory with space for five elements of type `int`.



The first element pointed by bobby can be accessed either with the expression bobby[0] or the expression *bobby. Both are equivalent as has been explained in the section about pointers. The second element can be accessed either with bobby[1] or *(bobby+1) and so on...

You could be wondering the difference between declaring a normal array and assigning dynamic memory to a pointer, as we have just done. The most important difference is that the size of an array has to be a constant value, which limits its size to what we decide at the moment of designing the program, before its execution, whereas the dynamic memory allocation allows us to assign memory during the execution of the program (runtime) using any variable or constant value as its size.

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, it is important to have some mechanism to check if our request to allocate memory was successful or not.

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

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.

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

```
bobby = new int [5];

    // if it 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.

This method can be specified by using a special object called nothrow as parameter for new:

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

In this case, if the allocation of this block of memory failed, the failure could be detected by checking if bobby took a null pointer value:

```
int * bobby;
bobby = new (nothrow) int [5];
if (bobby == 0) {
  // error assigning memory. Take measures.
  };
```

This `nothrow` method requires more work than the exception method, since the value returned has to be checked after each and every memory allocation, but I will use it in our examples due to its simplicity. Anyway this method can become tedious for larger projects, where the exception method is generally preferred. The exception method will be explained in detail later in this tutorial.

Operator delete and delete[]

Since the necessity of dynamic memory is usually limited to specific moments within a program, once it is no longer needed it should be freed so that the memory becomes available again for other requests of dynamic memory. This is the purpose of the operator `delete`, whose format is:

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

The first expression should be used to delete memory allocated for a single element, and the second one for memory allocated for arrays of elements.

The value passed as argument to delete must 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).

```
// rememb-o-matic
#include <iostream>
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 == 0)
    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,
```

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

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

But the user could have entered a value for i so big that our system could not handle 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). Remember that in the case that we tried to allocate the memory without specifying the nothrow parameter in the new expression, an exception would be thrown, which if it's not handled terminates the program.

It is a good practice to always check if a dynamic memory block was successfully allocated. Therefore, if you use the nothrow method, you should always check the value of the pointer returned. Otherwise, use the exception method, even if you do not handle the exception. This way, the program will terminate at that point without causing the unexpected results of continuing executing a code that assumes a block of memory to have been allocated when in fact it has not.

Dynamic memory in ANSI-C

Operators new and delete are exclusive of C++. They are not available in the C language. But using pure C language, dynamic memory can also be used through the functions malloc, calloc, realloc and free, defined in the <cstdlib> header file, and since C++ is a superset of C, these functions are also available to C++ programmers (see cstdlib for more info).

The memory blocks allocated by these functions are not necessarily compatible with those returned by new, so

each one should be manipulated with its own set of functions or operators.