C++

Dynamic Memory Allocation
A brief Cheat Sheet for Beginners

JANUARY 2023

Created by
Facundo Martínez
https://linkedin.com/in/fxmartinez

What is dynamic memory allocation?

There are two different types of memory allocation used by an application, known as "Heap" memory allocation and "Stack" memory allocation.

Stack allocation happens on contiguous blocks of memory, its size is fixed and allocation/deallocation happens automatically.

On the other hand, heap allocation takes place during the execution of instructions written by the programmer. Its size is much larger when compared to stack memory and it's accessible until the program stops being executed.

Is there any risk related to Heap allocation?

In C++, deallocation does not happen automatically (C++ does not have garbage collection), so deallocation must take place manually, unless we're using smart pointers.

Facundo Martínez
https://linkedin.com/in/fxmartinez

"new" and "delete" operators

```cpp
int main()
{
    // Declare pointer variable
    // and allocate it on heap
    // while initializing it
    int* p = new int(10);

    // Print the variable
    std::cout << p;
    // Console prints "10"

    // Release allocated memory
    delete p;

    return 0;
}
```

General syntax:

pointer-variable = new variable-type;

delete pointer-variable;

Facundo Martínez

https://linkedin.com/in/fxmartinez

Smart pointers: unique_ptr

Smart pointers are useful for automatic memory management, since they're automatically deleted when the program stops being executed and therefore avoid potential memory leaks.

unique_ptr is a type of smart pointer that stores one and only one pointer. You cannot point to another object unless you remove the first object from the pointer (using the move() method). The general syntax for this smart pointer is:

```
unique_ptr<Class-type> pointer-variable(new Class-type());

pointer-variable2 = move(pointer-variable);
```

Facundo Martínez

https://linkedin.com/in/fxmartinez

Smart pointers: unique_ptr

```cpp
#include <iostream>
#include <memory>

class Rectangle {
private:
    int length;
    int width;
public:
    Rectangle(int l, int w) {
        length = l;
        width = w;
    }
    int area() {
        return length * width;
    }
};

int main()
{
    // Using smart pointer to initialize
    // a Rectangle object
    unique_ptr<Rectangle> P(new Rectangle(3, 5));

    // Call method
    std::cout << P->area();

    return 0;
}
```

Facundo Martínez

https://linkedin.com/in/fxmartinez

Smart pointers: shared_ptr

Objects that were created using this pointer can be pointed at by multiple pointers (that's why it's called "shared pointer"). These pointers also carry a reference counter, to indicate how many pointers are pointing at this object.

```
int main()
{
    // Create object
    shared_ptr<Rectangle> P(new Rectangle(3, 5));
    // Print its area
    std::cout << P->area();         // Out: 15

    // Declare second shared_ptr
    shared_ptr<Rectangle> P2;
    P2 = P1;

    // Call area() method using P2
    std::cout << P2->area();        // Out: 15

    return 0;
}
```

Facundo Martínez

https://linkedin.com/in/fxmartinez

Smart pointers: weak_ptr

Weak pointers work like shared pointers, except that they do not hold a reference count.

```
int main()
{
    // Create object using weak pointer
    weak_ptr<Rectangle> P(new Rectangle(1, 2));

    return 0;
}
```

Facundo Martínez

https://linkedin.com/in/fxmartinez

More C++ resources at:

- C++ Reference
  https://cppreference.com
- W3 Schools
  https://w3schools.com/cpp
- Microsoft C++ documentation
  https://learn.microsoft.com/en-us/cpp/
- GeeksForGeeks
  https://geeksforgeeks.org/c-plus-plus

Facundo Martínez

https://linkedin.com/in/fxmartinez