History

Let's take a look at how C++ has evolved through the years.

WE'LL COVER THE FOLLOWING

C++98

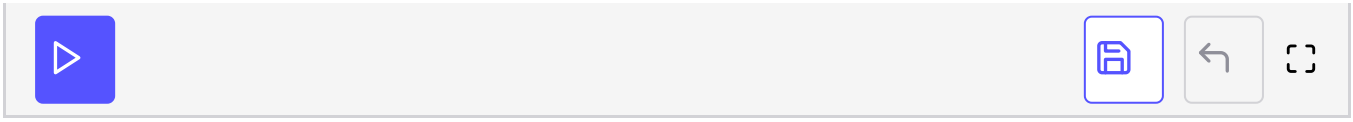
Work on C++ began back in 1985. However, the language was officially standardized in 1998. Hence, the original version was known as C++98.

Although its functionality was primitive compared to what we have now, C++98 supported basic I/O streams and templates. The standard library (STL) contained Strings, algorithms, and containers (arrays, queues, stacks, etc.), among other things.

Whereas C is a structured and procedural language, C++ was initially more focused on object-oriented programming:

```
#include <iostream>

class HumanBeing{
public:
  HumanBeing(std::string n): name(n){}
  std::string getName() const{
      return name;
  }
  void chanName(const std::string& newName){
    name = newName;
  }
private:
  std::string name;
};
```

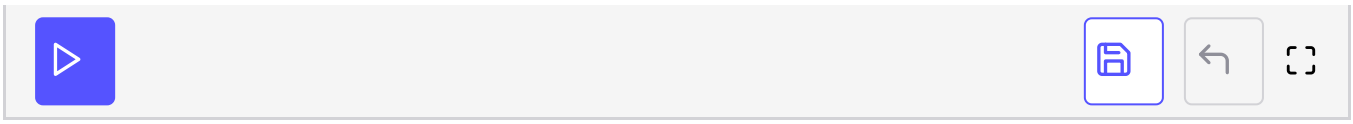
```
class Man: public HumanBeing{ using HumanBeing::HumanBeing;
public:
  std::string getSex() const{
     return "male";
  }
};

class Woman: public HumanBeing{ using HumanBeing::HumanBeing;
public:
  std::string getSex() const{
    return "female";
  }
};

int main(){
  Man man("Rainer");
  std::cout << "Name: " << man.getName() << std::endl;
  man.chanName("Rainer Grimm");
  std::cout << "New name: " << man.getName() << std::endl;
  std::cout << "Sex: " << man.getSex() << std::endl;
}
```



C++ also adopted a generic nature with the use of templates:

```
template <typename T>
void xchg(T& x, T& y){
  T t= x;
  x= y;
  y= t;
};

template <typename T, int N>
class Array{
public:
  int getSize() const{
    return N;
  }
private:
  T *elem[N];
};

int main(){
  // Using the first template
  int i = 10;
  int j = 20;
  std::cout << "i = " << i << ", j = " << j << std::endl;

  xchg(i, j);
  std::cout << "i = " << i << ", j = " << j << std::endl;

  Man huber("Huber"); // The Man class has already been created
  Man maier("Maier");
  std::cout << huber.getName() << ", " << maier.getName() << std::endl;
  xchg(huber, maier);
  std::cout << huber.getName() << ", " << maier.getName() << std::endl;
```

```
  // Using the second template
  Array<double, 10> doubleArray;

  std::cout << doubleArray.getSize() << std::endl;

  Array<Man, 5> manArray;
  std::cout << manArray.getSize() << std::endl;
}
```



C++11

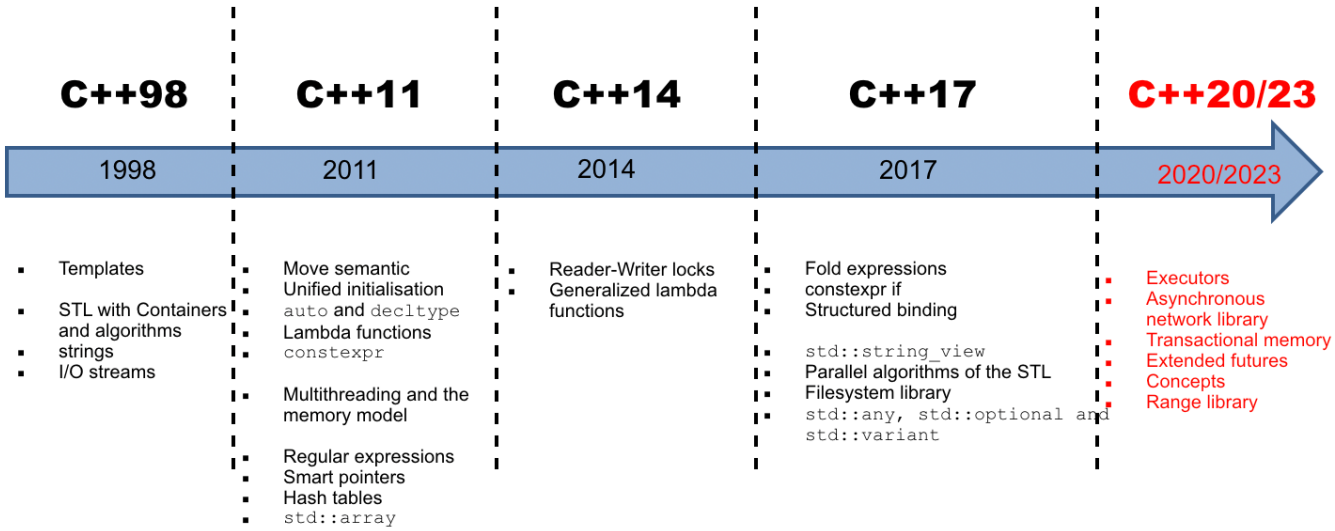
With the introduction of C++11, the language shifted its focus to **functional programming**. There were a lot of new features that, in a sense, evolved the language. Some of them can be seen below:

- Shared and unique pointers
- Null pointers
- Automatic type deduction
- Lambda expressions

Here's an example of a lambda function and automatic type deduction:

```
#include <iostream>
#include <vector>
#include <numeric>
#include <algorithm>
#include <functional>

int main(){
  std::vector<int> vec{1, 2, 3, 4, 5, 6, 7, 8, 9};
  std::vector<std::string> str{"Programming", "in", "a", "functional", "style."};

  std::accumulate(vec.begin(), vec.end(), [](int a, int b){return a*b;}); // 362880

  std::accumulate(str.begin(), str.end(),
                [](std::string a, std::string b){return a + ":"+ b;});
                // "Programming:in:a:functional:style."

  std::transform(vec.begin(), vec.end(), vec.begin(),
                 [](int i){ return i*i; });   // {1, 4, 9, 16, 25, 36, 49, 64, 81}

  auto it= std::remove_if(vec.begin(),vec.end(),
                 [](int i){ return ((i < 3) || (i > 8)); });  // {3, 4, 5, 6, 7, 8}

  auto it2= std::remove_if(str.begin(), str.end(),
                 [](std::string s){ return (std::tolower(s[0])); }); // "Programming"
}
```

Timeline



Further information

- Shared pointers
- unique pointers
- Null pointers
- Automatic type deduction
- Lambda expressions

In the next chapter, we will learn how to work with **literals**.