

by "Bjarne Stroustrup" in early 1980s

C++ [procedural and OOPS language]

efficiency

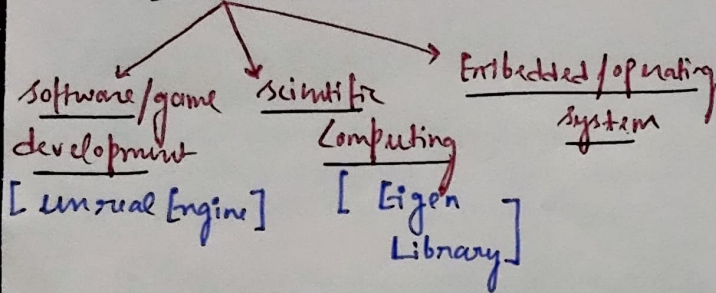
"Very close to hardware"

performance

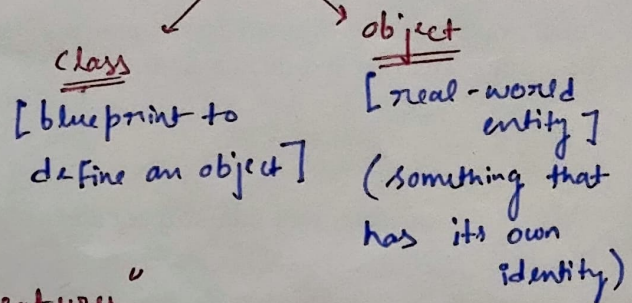
Extensive standard library

⇒ Ability to combine low-level memory manipulation with high-level abstraction.  
(concept of pointers & references) (OOPS support)

C++ use cases



OOPS { object oriented programming system }



"OOPS 4 pillars/features"

ENCAPSULATION

(data + function packed into single unit)

eg. class Animal {  
name;  
void walk();  
};

ABSTRACTION

(hiding complex details)  
using visibility labels

eg. class Animal {  
private:  
int frequency;  
};

INHERITANCE

"Re-usability"  
class Acquiring properties of another class

eg. class parent {  
protected:  
string name;  
};  
class child : public parent {  
int age;  
};

treated as private member of child

POLYMORPHISM

"many forms"

Base class Attribute Labels

child class visibility labels in Inheritance

public

public

private

protected

private

can't Access directly

can't Access directly

can't Access directly

protected

protected

private

protected



# Polymorphism [many forms] -

\* Function overriding: Same Name Function defined in parent and child class;

Compile time

Run-time

Func<sup>n</sup> overload  
[Same name Functions but differ in parameter types and number]

operator overload

"giving specific functionality to an operator, Only Applicable on user defined datatypes"

eg → class Complex {

real, imag;  
public:

```
complex operator + (complex const &obj) {
    complex result;
    result.real = real + obj.real;
    result.imag = imag + obj.imag;
    return res;
}
```

}

main() { complex c3 = c1 + c2; }

generic programming. Template <typename T>  
class demo {  
main() { demo<int> obj; }

passed as argument

operate on this

## Compile time polymorphism

Base class object can only call the function belonging to its class not CHILD class.

Because this has been a "static linkage" made by Compiler at Compile time

\* So that's why we use virtual functions such that a base class pointer can refer to both objects belonging to base class as well as child class

```
class shape {
public:
    virtual void getarea() {
        cout << "in shape";
    }
};
```

```
class circle: public shape {
public:
    void getarea() {
        cout << "in Circle";
    }
};
```

\* this linkage of call is made at run time that's why it's called run time polymorphism.

```
main() {
    shape *s1;
    circle c1(10);
    s1 = &c1;
    s1->getarea(); // in Circle
    s1 = new shape(10);
    s1->getarea(); // in shape
}
```

\* policy based data structure: Just like user defined data structure that are Flexible and used for special purpose / business purpose.

\* Bit set library: #include <bitset>, bitset<32>(7); // create a bitmap

run Command → g++ filename.cpp -o sample.exe  
/sample

31st bit ..... 0011  
0th

Exception handling → "handling unwanted events that occur during run-time of program"

```
try {
    // if (error) {
        throw "error";
    }
    catch (string exception) {
        // if type is Not
        // known use ...
    }
```