Programming Languages and Parameter Passing

By : Brady Davis

Types of Programming Languages

Declarative Languages:

- Logic Programming
- Functional Programming
- Database Driven Programming

Imperative Languages:

- Object-Oriented Programming
- Parallel Programming
- Procedural Programming

https://medium.com/javarevisited/70-years-of-hello-world-with-50-programming-languages-2400de893a97

Functional Programming

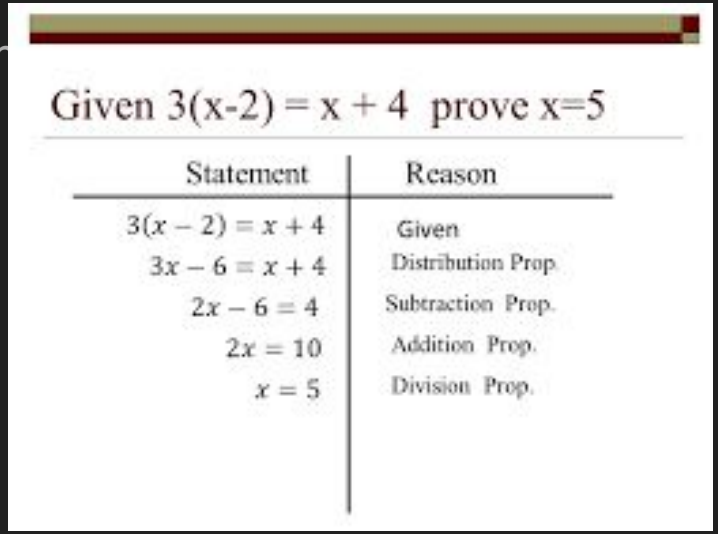
```
let meetups = [
  {name:'JavaScript', isActive:true, members:700},
  {name:'Angular', isActive:true, members:900},
  {name:'Node', isActive:false, members:600},
  {name:'React', isActive:true, members:500}
];
let sumFPChain = meetups.filter((m)=>{
    return m.isActive;
  })
  .map((m)=>{
    return m.members- (0.1*m.members);
  })
  .reduce((acc, m)=>{
    return acc + m;
  },0);
console.log(sumFPChain);   // Output will be 1890
```

https://codeburst.io/functional-programming-in-javascript-e57e7e28c0e5

- Focuses on “what to solve”
- Uses expressions instead of statements
- Concepts:
  - Pure Functions
  - Recursion
- Example Languages:
  - Lisp
  - Racket
  - JavaScript
  - Haskell

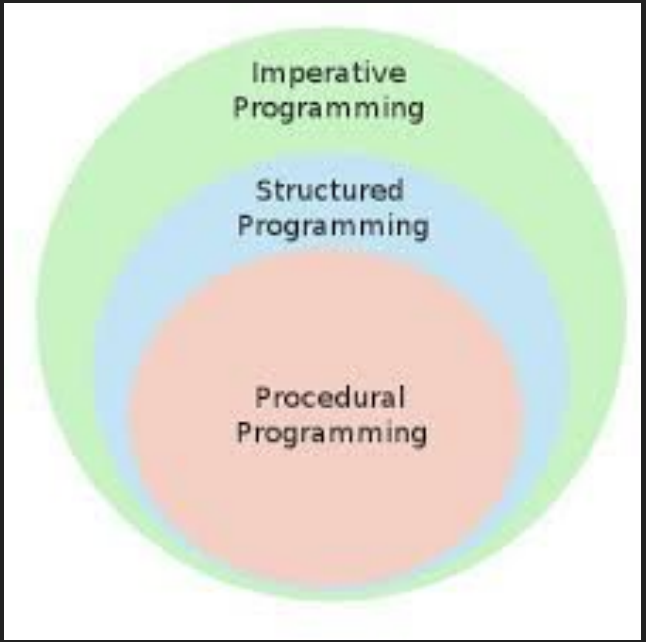
Logical Programming

- Emphasis on knowledge base and the problem
- Like a proof of mathematical statements
- Example Languages
  - Frill
  - Prolog
  - Mercury
  - ROOP



https://slideplayer.com/slide/7927700/

Imperative Programming

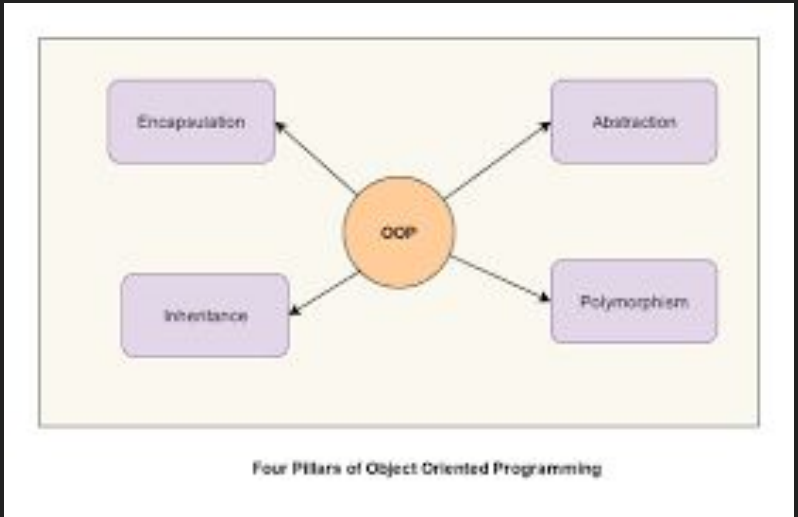


- One of the oldest programming paradigms
- Focuses on “how to achieve”
- Contains loops and variables
- Split into 3 categories
  - Procedural
  - Object-Oriented
  - Parallel Processing

https://softwareengineering.stackexchange.com/questions/117092/whats-the-difference-between-imperative-procedural-and-structured-programming

Object-Oriented Programming

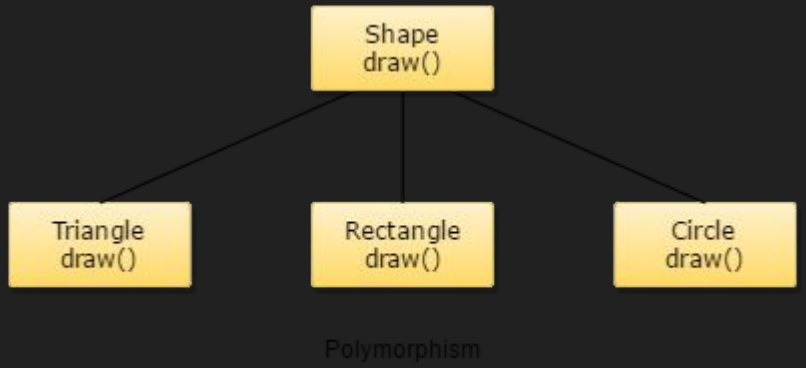
- More emphasis on data than procedure
- Handles real life problems with objects
- Characteristics
  - Polymorphism
  - Inheritance
  - Abstraction
  - Encapsulation



https://medium.com/@cancerian0684/what-are-four-basic-principles-of-object-oriented-programming-645af8b43727

Polymorphism

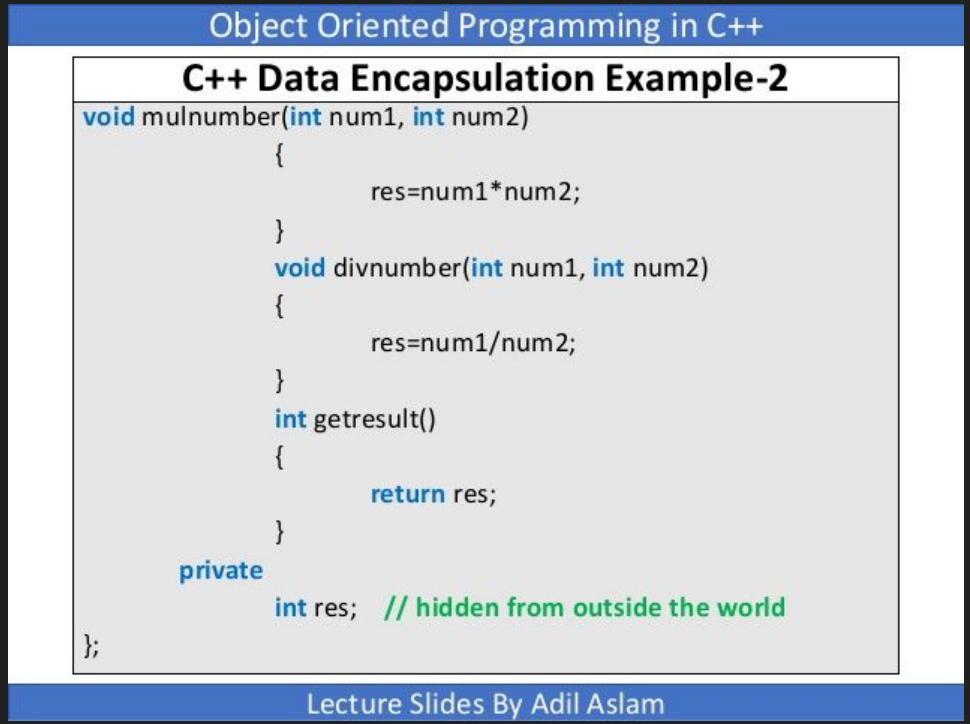
- Ability to differentiate between entities with the same name
- Function Overloading



Polymorphism

https://dzone.com/articles/learning-java-what-vs-why

Encapsulation

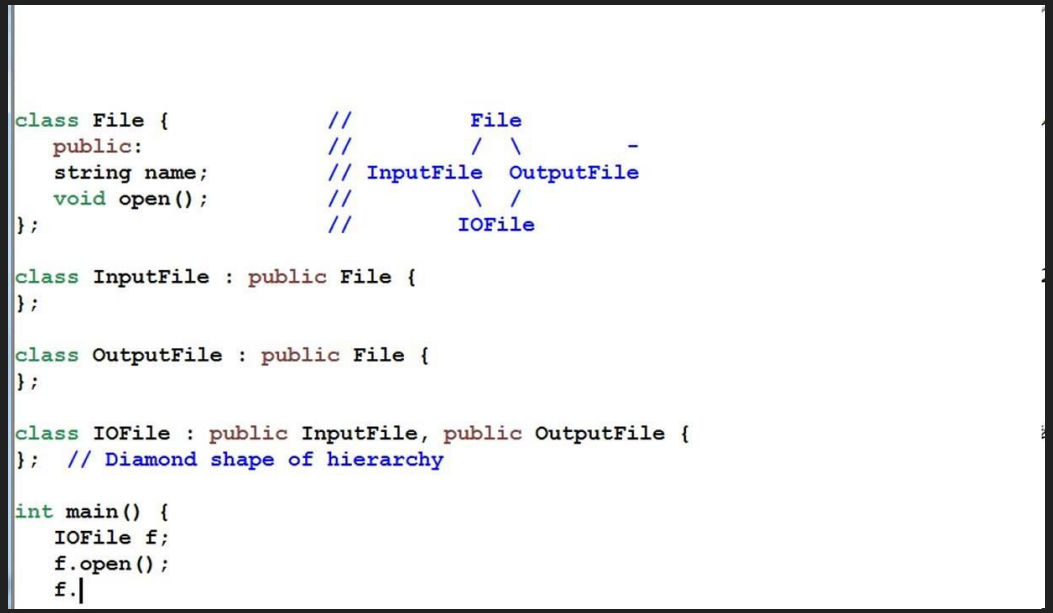


https://www.slideshare.net/AdilAslam4/object-oriented-programming-using-c-slides-44

- Protects variables from being accessed without the use of functions
- Also known as data hiding

Inheritance

- One class inherits the features of another class
- Key Terms
  - Super Class
  - Sub Class
  - Reusability



```
class File {            //          File
   public:              //          /  \        -
   string name;         // InputFile  OutputFile
   void open();         //          \  /
};                      //         IOFile

class InputFile : public File {
};

class OutputFile : public File {
};

class IOFile : public InputFile, public OutputFile {
};  // Diamond shape of hierarchy

int main() {
   IOFile f;
   f.open();
   f.
```

https://www.geeksforgeeks.org/inheritance-in-c/

Abstraction

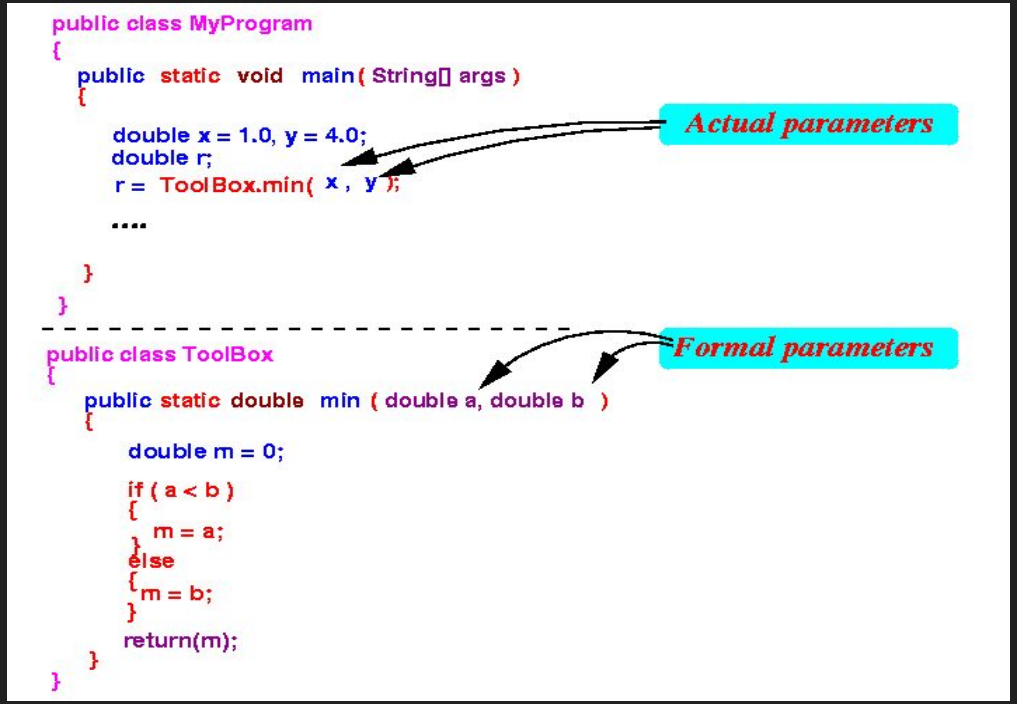
```
class X
{
public:
  X(int val) : data(val) {}
  void mf() { cout << "X::mf with value " << data << endl; }

private:
  int data;
};


int main()
{
  vector<X> v;

  v.push_back(X(100));
  v.push_back(X(101));
  v.push_back(X(102));

  // Now, let's iterate...

}
```

- Only the essential details are displayed to the user
- Similar to encapsulation, but not the same

https://blog.feabhas.com/tag/iteration/

Parameter Passing Techniques

- Pass by Value
- Pass by Pointer
- Pass by Reference



http://www.mathcs.emory.edu/~cheung/Courses/170/Syllabus/08/parameter2.html

Pass by Value

```
1  void Two (int x)
2  {
3      x = 2;
4      cout << x << endl;
5  }
6
7  void One ()
8  {
9      int y = 1;
10     Two (y);
11     cout << y << endl;
12 }
```

https://book.huihoo.com/data-structures-and-algorithms-with-object-oriented-design-patterns-in-c++/html/page592.html

- Parameters copied to function arguments
- If modifications are done the actual value does not change

Pass by Pointer

- Uses pointers
- If modifications are done, the actual value IS changed automatically

```
#include <iostream>

using namespace std;

void my_swap(int *x, int *y) {

      int temp;

      temp = *x;

      *x = *y;

      *y = temp;

}

int main() {

      int a, b;

      a = 10;

      b = 40;

      cout << "(a,b) = (" << a << ", " << b << ")\n";

      my_swap(&a, &b);

      cout << "(a,b) = (" << a << ", " << b << ")\n";

      }
```

https://www.tutorialspoint.com/parameter-passing-techniques-in-c-cplusplus

Pass by Reference

- Pass the reference variable of an argument
- The actual value IS automatically updated on modifications
- Only have to put ‘&’ at the function definition

```
#include <iostream>

using namespace std;

void my_swap(int &x, int &y) {

   int temp;

   temp = x;

   x = y;

   y = temp;

}

int main() {

   int a, b;

   a = 10;

   b = 40;

   cout << "(a,b) = (" << a << ", " << b << ")\n";

   my_swap(a, b);

  cout << "(a,b) = (" << a << ", " << b << ")\n";

}
```

https://www.tutorialspoint.com/parameter-passing-techniques-in-c-cplusplus

References

https://www.geeksforgeeks.org/object-oriented-programming-oops-concept-in-java/

https://www.geeksforgeeks.org/functional-programming-paradigm/

https://www.geeksforgeeks.org/introduction-of-programming-paradigms/

https://www.tutorialspoint.com/parameter-passing-techniques-in-c-cplusplus