



Problems and issues with Procedural Programming



Storing Data in Procedural Way

We have made following **Parallel arrays** to store records of the students.

```
int array_size = 5;  
string [] sname = new string[array_size];  
float [] matricMarks = new float[array_size];  
float [] fscMarks = new float[array_size];  
float [] ecatMarks = new float[array_size];  
float[] aggregate = new float[array_size];
```

Storing Data in Procedural Way

Can you highlight any problem with this approach ?

```
int array_size = 5;  
string [] sname = new string[array_size];  
float [] matricMarks = new float[array_size];  
float [] fscMarks = new float[array_size];  
float [] ecatMarks = new float[array_size];  
float[] aggregate = new float[array_size];
```



Disjoint Data

Record of a student is **highly correlated** but it is stored in different **disjoint arrays** and linked with the index number only.

```
int array_size = 5;  
string [] sname = new string[array_size];  
float [] matricMarks = new float[array_size];  
float [] fscMarks = new float[array_size];  
float [] ecatMarks = new float[array_size];  
float[] aggregate = new float[array_size];
```

Disjoint Data issues: Data Management

Data is distributed in **different arrays**. If you want to pass complete record of any student than you need to pass data from different arrays.

```
int array_size = 5;
string [] sname = new string[array_size];
float [] matricMarks = new float[array_size];
float [] fscMarks = new float[array_size];
float [] ecatMarks = new float[array_size];
float[] aggregate = new float[array_size];
aggregate[0] = calculateMerit(matricMarks[0], fscMarks[0], ecatMarks[0]);
```

Disjoint Data issues: Data Security

We can not **restrict** data access. For example, if we want a print function should not **access** a specific record in the Array then it is **not possible**.

```
static void printRecords(string[] name, float[] matric, float[] fsc, float[] ecat, float[] agg)
{
    for (int x = 0; x < 5; x++)
    {
        Console.WriteLine("{0} \t {1} \t {2} \t {3} \t {4}", name[x], matric[x], fsc[x], ecat[x], agg[x]);
    }
}
```

Disjoint Data issues: Constraint on Data

Data can not look after itself. We can not add any **constraint** on the manipulation or access of the Data.

```
int array_size = 5;  
string [] sname = new string[array_size];  
float [] matricMarks = new float[array_size];  
float [] fscMarks = new float[array_size];  
float [] ecatMarks = new float[array_size];  
float[] aggregate = new float[array_size];
```

Disjoint Data issues: Constraint on Data

For example, there could be no **guarantee** that in the `firstYearMarks` the value shall always less than **1100** and greater than **0**.

$0 < \text{firstYearMarks}[] < 1100$



```
int array_size = 5;  
string [] sname = new string[array_size];  
float [] matricMarks = new float[array_size];  
float [] fscMarks = new float[array_size];  
float [] ecatMarks = new float[array_size];  
float[] aggregate = new float[array_size];
```


Operations on Data

We define functions to perform operations on the Data but the function and data are highly decoupled.

```
aggregate[0] = calculateMerit(matricMarks[0], fscMarks[0], ecatMarks[0]);
```

```
static float calculateMerit(float mMarks, float fscMarks, float ecatMarks)
{
    float score;
    score = ((0.25F * (mMarks / 1050)) + (0.45F * (fscMarks / 550)) + (0.3F *
        (ecatMarks / 400))) * 100;
    return score;
}
```

Operations on Data: is it Problematic?

We define functions to perform operations on the Data but the function and data are highly decoupled.

```
aggregate[0] = calculateMerit(matricMarks[0], fscMarks[0], ecatMarks[0]);
```

```
static float calculateMerit(float mMarks, float fscMarks, float ecatMarks)
{
    float score;
    score = ((0.25F * (mMarks / 1050)) + (0.45F * (fscMarks / 550)) + (0.3F *
        (ecatMarks / 400))) * 100;
    return score;
}
```

Operations on Data: Maintainability

If any change occurs in the data, all other functions need to be changed but they are distributed all over the code and it is easy to miss any change that leads to the crash of the software.

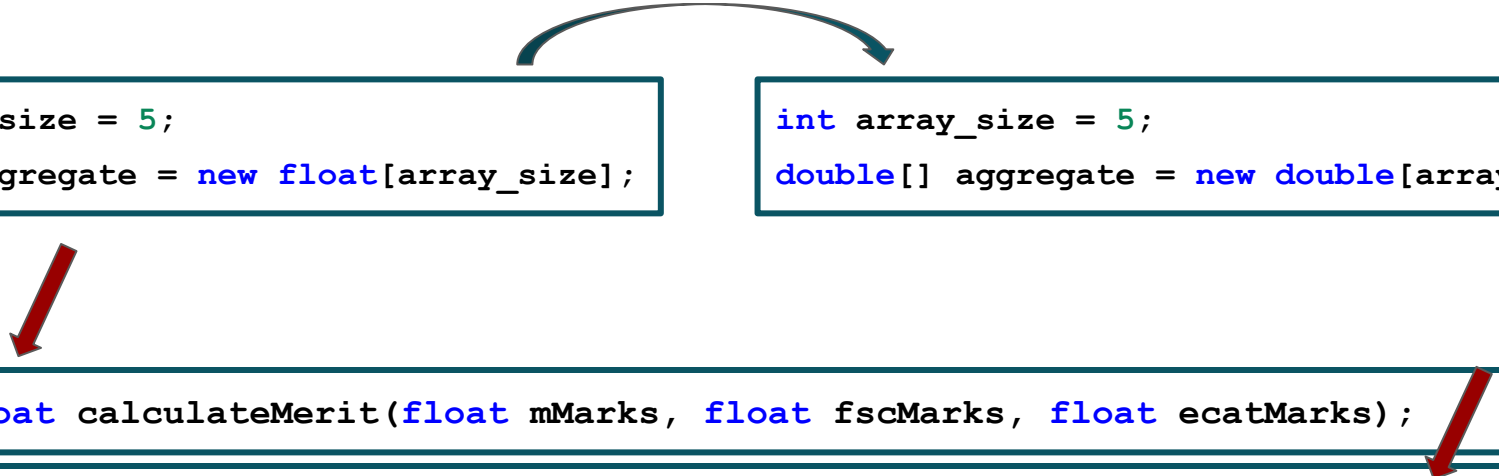
```
int array_size = 5;
string [] sname = new string[array_size];
float [] matricMarks = new float[array_size];
float [] fscMarks = new float[array_size];
float [] ecatMarks = new float[array_size];
float[] aggregate = new float[array_size];
```

```
static float calculateMerit(float mMarks, float fscMarks, float ecatMarks);
```

```
static void printRecords(string[] name, float[] matric, float[] fsc, float[] ecat, float[] agg);
```

Operations on Data: Maintainability

For example, we need to change the datatype of **aggregate** from **float** to **double**. Now this change require to keep changes at all place where the data has been referred. How to find all those places ?



```
int array_size = 5;  
float[] aggregate = new float[array_size];
```

```
int array_size = 5;  
double[] aggregate = new double[array_size];
```

```
static float calculateMerit(float mMarks, float fscMarks, float ecatMarks);
```

```
static void printRecords(string[] name, float[] matric, float[] fsc, float[] ecat, float[] agg);
```

Hiding the Details

In procedure programming, when **Program A** provide access to **Program B**, all features, functions and data get accessible to program B.



Hiding the Details

We do not have any way to provide **partial access** in procedural programming.

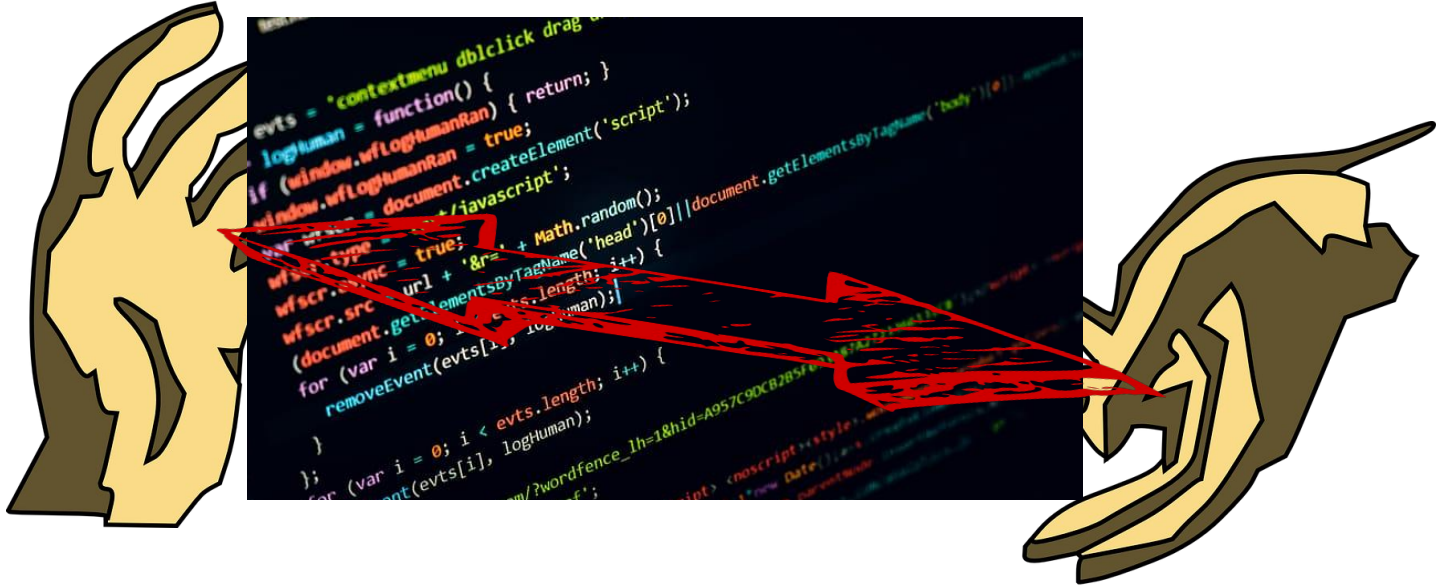


Extending the Functionality

Sometimes, Program A needs same functionality that program B has. Like, in case of aggregate calculations that may be needed in different programs such as admission program, scholarship program, hostel program and roll number assignment program.

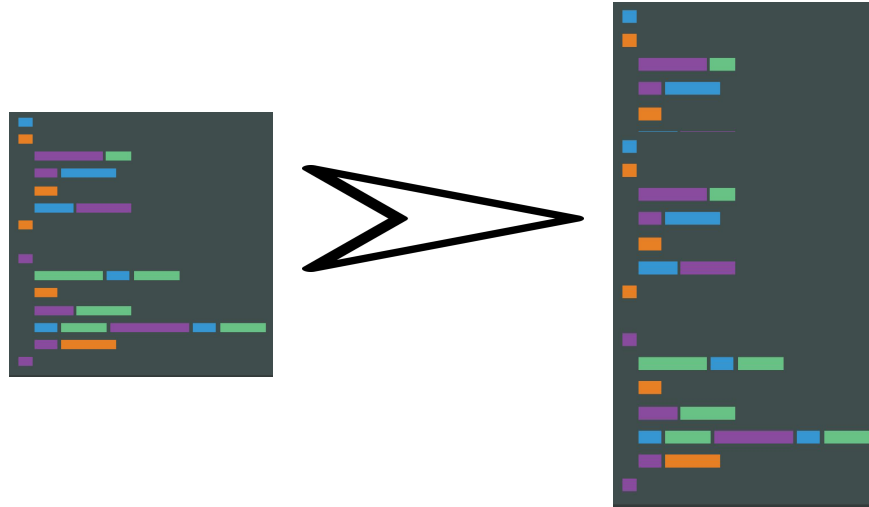
Extending the Functionality

Some times, We do not want to write a **complete program** again but we do want to **add** some new functionality.



Extending the Functionality

It is same like when we are calculating fee for a **day scholar** student but to calculate fee for a **hostelite student**, we just need to **add some extra calculation** into the program.



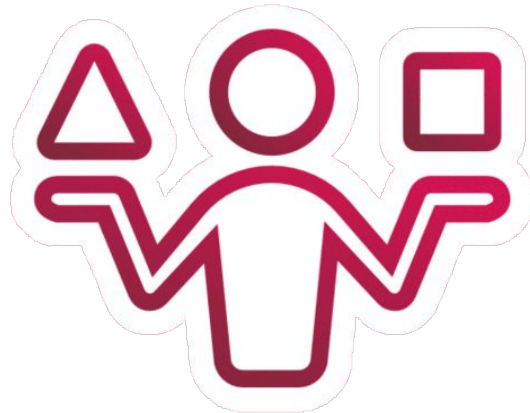
Extending the Functionality

In procedural programming, it is **not possible** or very difficult to **extend the functionality** of existing code.



Different Behaviour

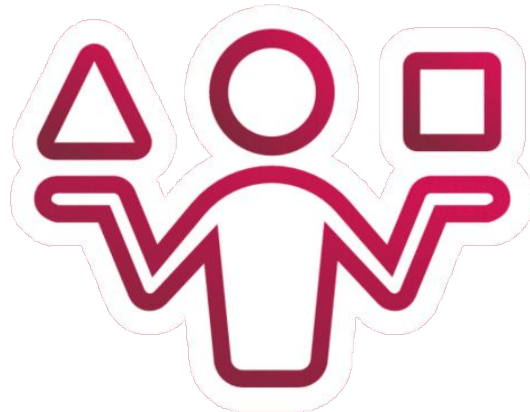
It is very much practical that we need **same function** behave **differently** for different scenarios.



Different Behaviour

For example, we know there should be a function `area()` to calculate **area of a shape**. This functional shall behave different for different type of shapes.

For example, for **triangle** its formula shall be different and for the **square** its formula shall be different.





Conclusion

Following are the **major limitations** of procedural programming

- NOT A **Single Unit** of both Data and Functions
 - Disjoint Data
 - Disjoint Behaviour
- Providing **Partial Access** to Part of Program.
- Extending the **Functionality** of the Program.
- Different Functions behave **differently** for different scenarios



Learning Objective

Explain the limitations of the
procedural programming.



Self Assessment

1. Identify the **code areas** in your previous projects that have issues due to these limitations.

- Identify at least **2 examples** of **Disjoint behaviour** from your business application and 2 examples from your Game
- Identify at least **2 examples** of copying the code instead of **extending the functionality** in both your Business application and Game project.
- Identify at least **2 examples** where you have given the **complete access** to arrays whereas only **partial access** is required.

