



**MUST**  

---

**Wisdom & Virtue**

MIRPUR UNIVERSITY OF SCIENCE AND TECHNOLOGY (MUST)  
DEPARTMENT OF SOFTWARE ENGINEERING



# Object Oriented Programming

## Lecture 1: Introduction to OOP

*Engr. Saman Fatima*

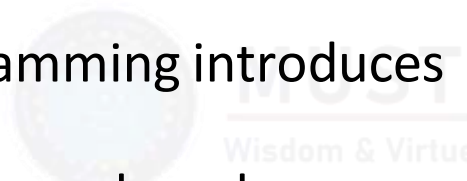


# COURSE DESCRIPTION

After covering basic programming skills. It is very much necessary to introduce a programming paradigm which is very much powerful as compared to basic programming techniques.

In this regard object-oriented programming introduces

- Real world programming concepts and modern ways to handle real world problems
- The benefit of this course is students can solve more complex problems with latest guidelines and skills

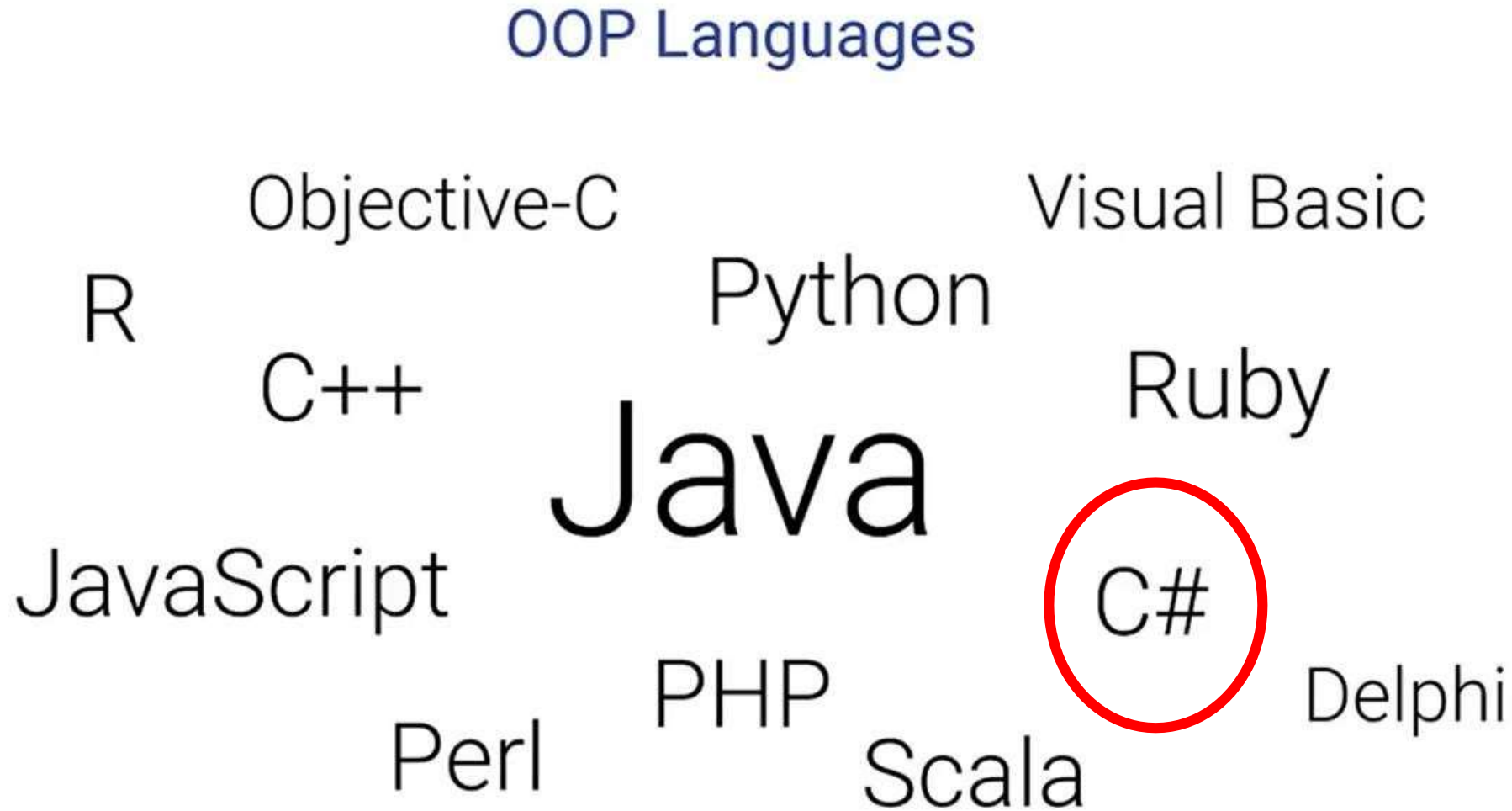


# COURSE OBJECTIVES

*Objective of this course are:*

- Describe Object Oriented Programming
- Tackle real world problems through Object Oriented Programming
- Handle Operator overloading
- Implement Inheritance, Polymorphism, Encapsulation and related features of OOP
- Handle Constructors & Destructors
- Implement Generic Programming Concepts and manipulate exception handling

# Tool



# COURSE TEXTBOOKS

- Object Oriented Programming in C++ Robert Lafore
- Beginning C# Object Oriented Programming by Dan Clark
- C# How to Program by Dietel & Dietel.
- Beginning Object Oriented Programming with C# by Jack Purdum



# GRADING POLICY

***Mid Term:*** [30%]

***Quizzes:*** [10%]

***Assignments:*** [10%]

***Final Exam:*** [50%]



# HOMework & ASSIGNMENT POLICY

- There will be 2 assignments and two quizzes in this courses.

Assignment 1	3 <sup>rd</sup> WEEK
Assignment 2	6 <sup>TH</sup> WEEK
Quiz 1	After 4 <sup>th</sup> Week
Quiz 2	After 12 <sup>th</sup> Week

- Policies
  - Late assignments may be accepted with marks reduction. There will be a 10% reduction for assignments submitted up to 24 hours late and 100% after that.
  - Students who have copied assignments or whose assignments have been copied will both be given a zero
  - Plagiarism is not acceptable. Anyone found to be guilty of plagiarism in an assignment will be given a zero in that assignment
  - Quizzes may be unannounced or announced (depends on your response!)





# Lecture Contents

1. Object Oriented Programming Misconceptions

2. Problems with Procedural Paradigm



# Object Oriented Programming Misconceptions

**Question1:** Am I going to learn a new programming language?

***No OOP is not a programming Language.***

**Question2:** If Not language then what is it?

*OOP is a “technique” to manage your code. Such that it provide you many benefits. (**what??**)*



# Object Oriented Programming Misconceptions

## Question3: What do you mean by technique?

In the last semester you have studied Fundamental of programming.

You learn about programming language fundamentals

- Variables, loops, functions, struct and pointers etc. By using these you built your programs. Like calculator which provide addition/ subtraction / division/multiplication.
- Simple Calculator has let say two variables and 4 functions.
- But these programs are manageable in a single file. You call it calculator.cs.

- ❑ Now what happens if your program size grow.
- ❑ Let say 100 different functions and 100 different variables.
- ❑ So, you need some technique so that you manage your code in a way which is easy to “**debug**” and “**maintain**” and “**reuse**”.



# Reusability

- Let say, you are working on a project, and it is required to developed a simple calculator program and you save it in Calculator.cs
- Now in the same project, a scientific calculator is also required. Which has all functionality of simple calculator and plus other functionality.
- There is no way to re-use simple calculator code except you must copy/paste it in scientific calculator code file. Its mean you had duplicated your code.
- So, concept of re-usability of code is to use code without duplication.
  - Think like that! you write a code in some program, and you use it in any other program without copy paste. Don't you think it is supper fun.

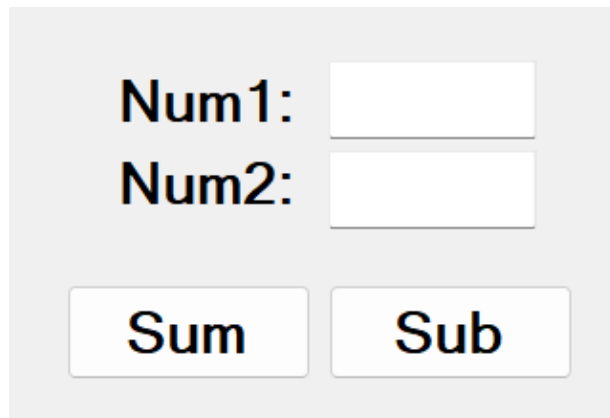


# Understand the Concept of Reusability with a Simple Example



# Simple Calculator with two Basic Functions

- We are required to create a simple calculator with two basic functions in a project



Num1:

Num2:

Sum Sub

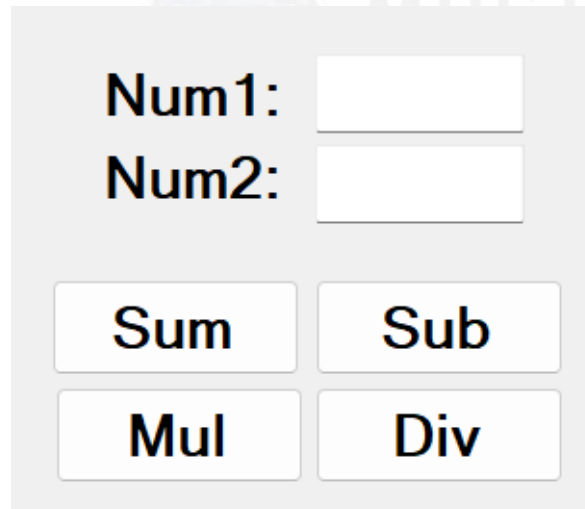


```
double sum(double num1, double num2)
{
    return num1 + num2;
}
double sub(double num1, double num2)
{
    return num1 - num2;
}
```



# Simple Calculator with two Basic Functions

- Later, in the same project another calculator with four basic functions is required

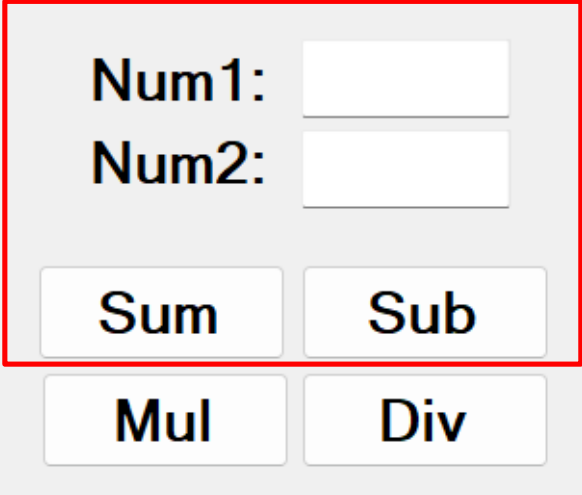


A simple calculator interface with two input fields and four operation buttons. The interface is displayed on a light gray background. The first input field is labeled "Num1:" and the second is labeled "Num2:". Below the input fields are four buttons arranged in a 2x2 grid: "Sum", "Sub", "Mul", and "Div".



# Simple Calculator with two Basic Functions

- 90% function is already implemented in simple calculator with two basic functions.
- Since we are working in procedural programming, hence we have no other way except to copy and paste code and user interface (**No Reusability**).

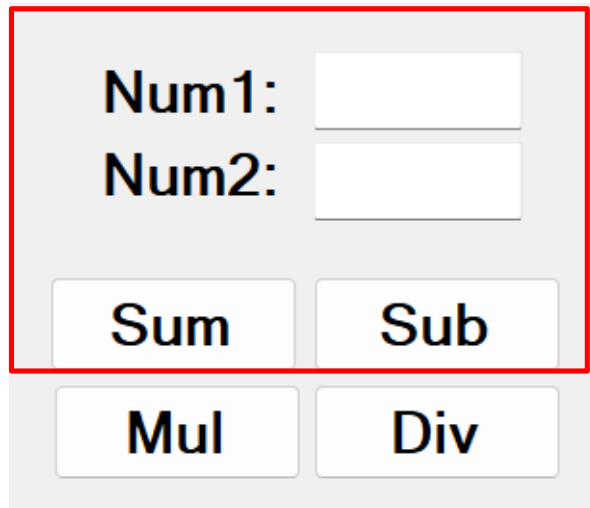


The image shows a simple calculator user interface. It features two input fields labeled 'Num1:' and 'Num2:'. Below these fields are four buttons arranged in a 2x2 grid: 'Sum', 'Sub', 'Mul', and 'Div'. The entire calculator interface is enclosed in a red rectangular border.





# Simple Calculator with two Basic Functions



Num1:

Num2:

Sum Sub

Mul Div

```
double sum(double num1, double num2)
{
    return num1 + num2;
}
double sub(double num1, double num2)
{
    return num1 - num2;
}
```

```
double mul(double num1, double num2)
{
    return num1 * num2;
}
double div(double num1, double num2)
{
    return num1/num2;
}
```



# History towards OOP

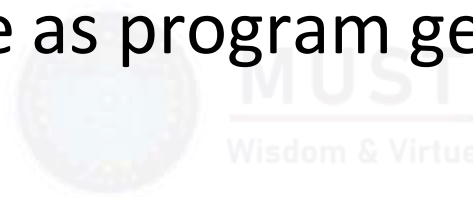
Let us understand road programmers travelled towards OOP technique

- Unstructured programming
- Procedural programming
- Structured programming
- Object-oriented programming



# Unstructured Programming

- Simple / small application consisting of **one main program**
- Program = sequence of commands (statements) which modify global data
- Drawback: unmanageable as program gets bigger; a lot of copy/paste-ed code



# Problem with Unstructured Programming



Inefficient Code



Spaghetti Code

[www.ezed.in](http://www.ezed.in)

# Procedural Programming

- A **procedural language** is a computer programming language that follows, in order, a set of commands.
- A procedural programming language consists of a set of **procedure calls(Functions)** and a set of code for each procedure.
- They make use of **functions**, **conditional statements**, and **variables** to create programs that allow a computer to calculate and display a desired output.
- Examples of procedural language include COBOL, LISP, Perl, HTML, VBScript


# Example Procedural Programming Code

- Here is the area of a triangle again, in the language C

```
int main(int argc, char *argv[])
{
    int a, b, c;
    printf("\nSide A: ");
    scanf("%d", &a);
    printf("\nSide B: ");
    scanf("%d", &b);
    printf("\nSide C: ");
    scanf("%d", &c);
    printf("\nThe area of the triangle is: %f\n", area (a, b, c));
    return(0);
}

float area ( int a, int b, int c )
{
    int s;
    float y;
    s = (a + b + c)/2;
    y = sqrt( s * (s - a)*(s - b)*(s - c) );
    return( y );
}
```

Here is where the procedure  
is called



Here is the procedure for computing the  
area of a triangle. Though called only once,  
it could be called from many places in the  
code, and could easily be reused in another  
program.

Note that variables a, b, c in the procedure  
are *different* variables (but, since they are  
passed to the procedure, the same values)  
as the variables a,b,c in the main program.

# Problems Procedural Programming

- The program code is **harder to write** when Procedural Programming is employed
- The Procedural code is often **not reusable**, which may pose the need to recreate the code if is needed to use in another application
- Difficult to relate with real-world objects
- The data is exposed to the whole program, leading to **unsafe code**.

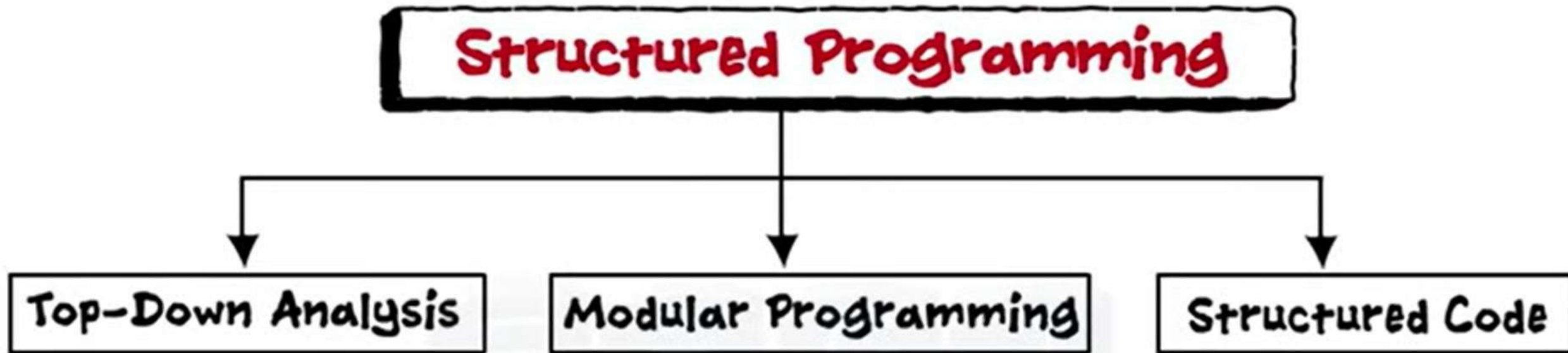
# Structured Programming

- Structured programming encourages dividing an application program into a **hierarchy of modules** or autonomous elements,
- Which may, in turn, contain other such elements.
- Within each element, code may be further structured using blocks of related logic designed to improve readability and maintainability.
- These may include case, which **tests a variable against a set of values; Repeat, while and for, which construct loops** that continue until a condition is met.





# Structured Programming



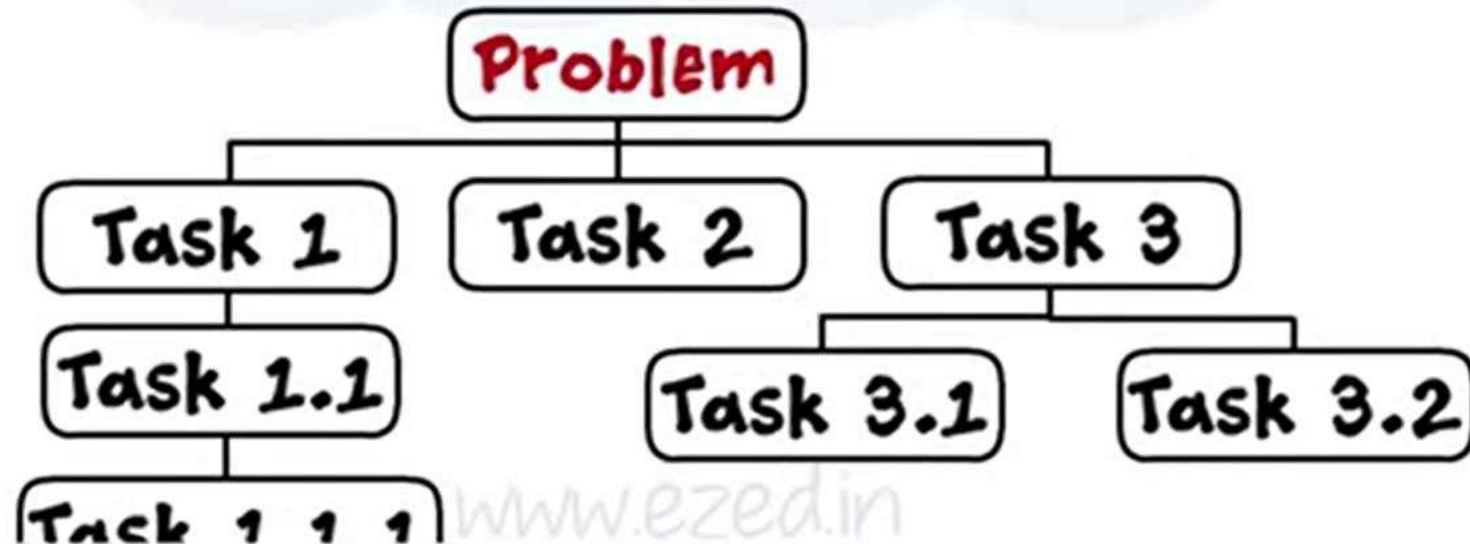
# Top-down Analysis

## Ideas

- Subdivision of Program

P R O B L E M

- Hierarchy of tasks



# Modular Programming

- Program designing style

```
main()  
{  
.....  
}
```

independent

```
.....  
}  
module1  
{  
.....  
}
```

independent

```
.....  
}  
module2  
{  
.....  
}
```

independent

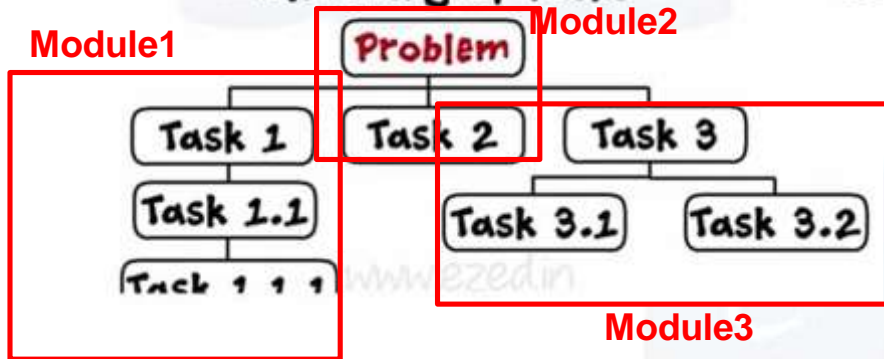
```
.....  
}  
.....  
}
```



# Modular Programming

## ● Program designing style

- Hierarchy of tasks



```
main()
{
```

```
.....
```

independent

```
.....
```

```
}  
module1
```

independent

```
.....
```

```
.....
```

```
}  
module2
```

independent

```
.....
```

```
.....
```

```
}  
www.ezed.in
```



# Modular Programming

module 1



module 2



module 3



module 4



Multiple Programmers can work simultaneously





module 1



module 2




module 3



module 4

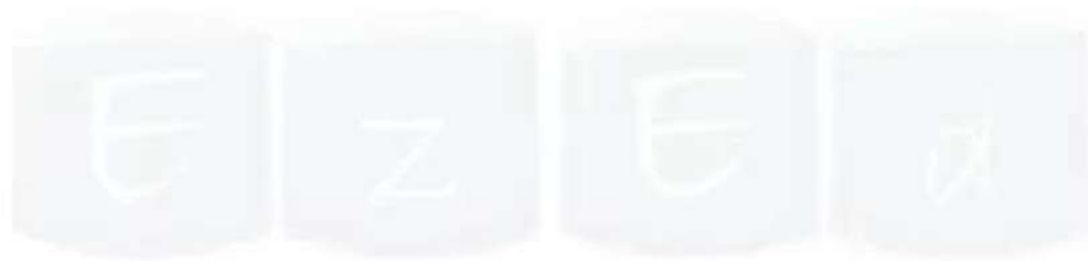


- It leads to reusable code

-  **easy** (Finding Errors is Easy)



# Structured Code



[www.ezed.in](http://www.ezed.in)



# Structured Code

Old Languages



`E goto d`

Unstructured Code





## if, if-else, else if

Structured Coding

www.ezed.in



# Structured Code

Normal flow of execution:

```
1 main()
2 {
3     .....
4     .....
5     .....
6 }
```

Altered flow of execution:

```
1 main()
2 {
3     ...
4     ...
5     if(condition)
6     {
7         do this
8     }
9 }
```



# Structured Code

Normal flow of execution:

```
1 main()  
2 {  
3 .....  
4 .....  
5 .....  
6 }
```

Altered flow of execution:

```
1 main()  
2 {  
3 ...  
4 ...  
5 if(condition)  
6 {  
7     do this  
8 }  
9 }
```

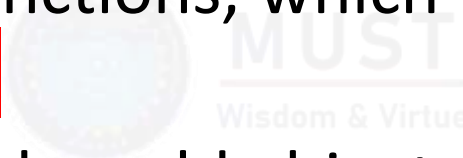
if, switch-case, loops like for, while, do-while

[www.ezed.in](http://www.ezed.in)




# Problems Structured Programming

- No matter how well the structured programming approach is implemented, large programs become **excessively complex**.
- Importance is given to functions, which makes programs **difficult to understand and manage**.
- Difficult to relate with real-world objects
- The data is exposed to the whole program, leading to **unsafe code**.



Object-oriented programming was developed because limitations were discovered in earlier approaches to programming.



To appreciate what OOP does, we need to understand what these limitations are and how they arose from traditional programming languages.

## Why Do We Need Object-Oriented Programming?



# Problems in Traditional Programming

## **Problems in Traditional Programming (Before OOP)**

### **Code Reusability Issue**

Every time you needed a similar function, you had to rewrite it instead of reusing code efficiently.

### **Difficult Maintenance**

If changes were needed, developers had to modify multiple parts of the code, which was time-consuming and error-prone.

### **Data Security**

In procedural programming, data was freely accessible by all parts of the program, leading to security risks.

### **Scalability Issue**

As programs became larger, maintaining and debugging them became more complex.



# OOP Solve These Problems

## Key Features of OOP:

**Encapsulation** → Data is hidden and accessed only through defined methods.

**Inheritance** → Code can be reused by creating new classes from existing ones.

**Polymorphism** → Functions and objects can take multiple forms, increasing flexibility.

**Abstraction** → Only the necessary details are shown, hiding complexity.



# Unrestricted Access

In procedural language, two kinds of data

- 1. Local Data**
- 2. Global Data**



# Local Data

- **Local Data (Remember Variable Scope in between {} these two Curley brackets)**
  - Local access to function
  - Safe from modification by other functions but you cannot access in other functions.
  - But if you want other functions to access these variables you make it global.

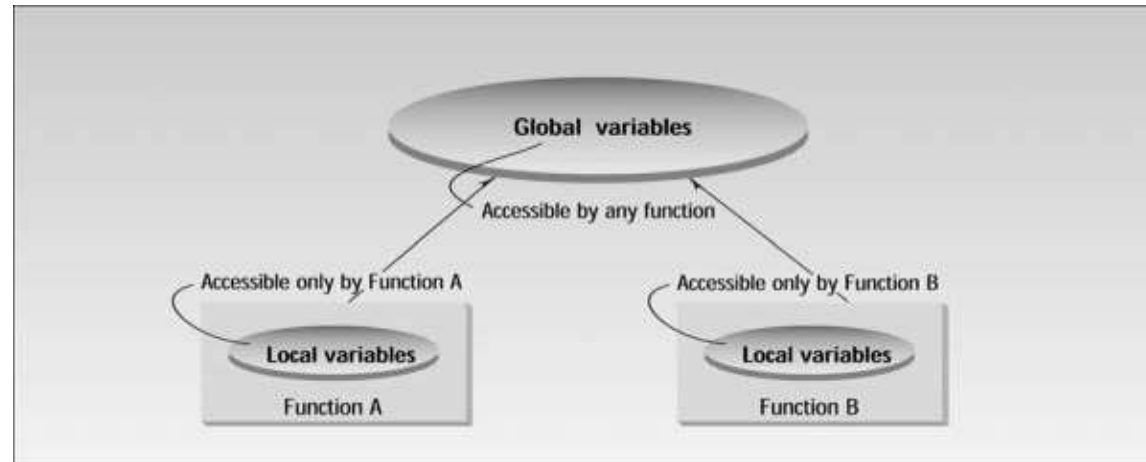
```
int AddNumber()  
{  
    int num1;  
    int num2;  
    int num3;  
    num3 = num1 + num2;  
    return num3;  
}
```



# Global Data

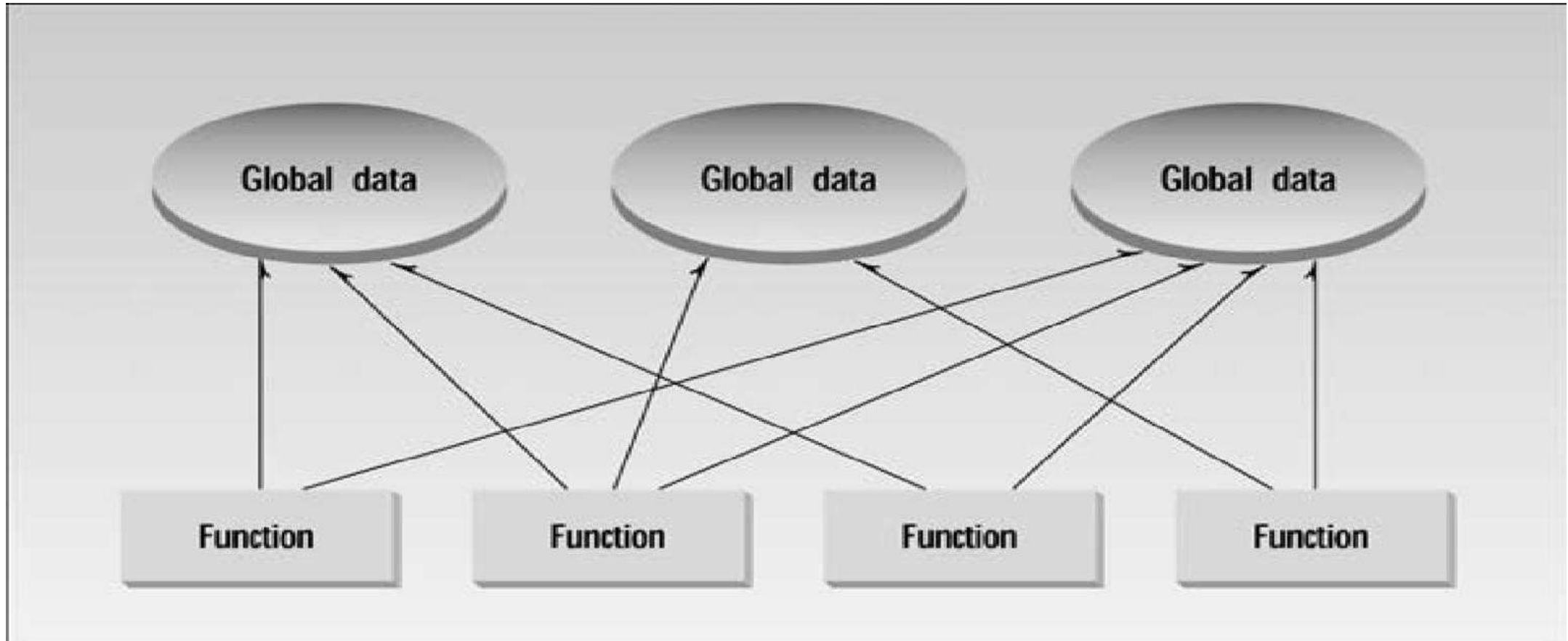
## ? Global data

- ? When two or more functions need to access same data then it is declared as **global data**.
- ? Global data can be accessed by any function in a program, you cannot hide data which leads- Difficult to modify/debug the program. e.g You never know which function change num3 value. (**Point noted: Data hiding**)



```
int num1=2;
int num2=5;
int num3;
Void AddNumber()
{
    num3 = num1 + num2;
}
Void SubNumber()
{
    num3 = num2 - num1;
}
```

# Cont'd



# Big Picture

- You understand the problems associated with past programming techniques.
  - First, functions have **unrestricted access** to global data.
  - Second, unrelated functions and data, the basis of the procedural paradigm, provide a **poor model of the real world**.
- To overcome these problems, Object Oriented Programming has been introduced

# References

- Object Oriented Programming in C++ Robert Lafore, Chapter 1.



THANKS