# Question Bank Answers (PCPF – IAE1)

CONTENT WARNING: READING THIS DOCUMENT MAY CAUSE SUDDEN BURSTS OF INTELLIGENCE. 7

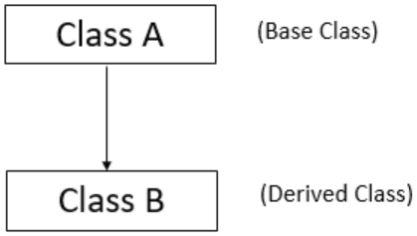
**1. What is Inheritance in OOP? Explain different types of Inheritance in OOP?**

**Ans.**

Inheritance in Object-Oriented Programming (OOP) allows a new class (called a subclass or derived class) to inherit attributes and methods from an existing class (called a superclass or base class). This promotes code re-usability.

**Types of Inheritance:**

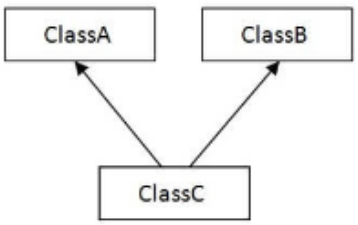
- **Single Inheritance:** A subclass inherits from one superclass.




**Syntax:**

```
class base_class_1
{
   // class definition
};
class derived_class: visibility_mode base_class_1
{
   // class definition
};
```

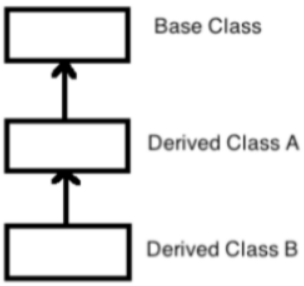
- **Multiple Inheritance:** A subclass inherits from more than one superclass (not supported directly in all languages, like Java).




**Syntax:**

```
class BaseClass1 {
   // Data members of BaseClass1.
   // Member functions of BaseClass1.
}
class BaseClass2 {
   // Data members of BaseClass2.
   // Member functions of BaseClass2.
}

// Multiple inheritance
class DerivedClass: public BaseClass1, public BaseClass2 {
   // Data members of DerivedClass.
   // Member functions of DerivedClass.
}
```

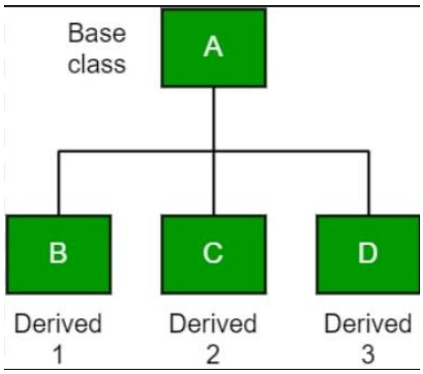
- **Multilevel Inheritance:** A subclass inherits from a superclass, and that superclass inherits from another class.




**Syntax:**

```
class A {
 // class definition
};
class B: public A {
 // class definition
};
class C: public B {
 // class definition
};
```

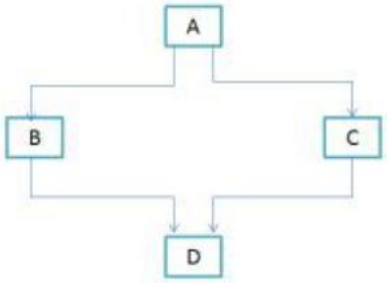
- **Hierarchical Inheritance:** Multiple subclasses inherit from a single superclass.




**Syntax:**

```
class base_class {
 // class definition
}
class first_derived_class: public base_class {
 // class definition
}
class second_derived_class: public base_class {
 // class definition
}
class third_derived_class: public base_class {
 // class definition
}
```

- **Hybrid Inheritance:** A combination of two or more types of inheritance (e.g., a mix of single and multiple inheritance).



(e) Hybrid Inheritance

**Syntax:**

```
Class A
{
        //Data members and member functions of Class A
};
Class B: access modifier A
{
        //Data members and member functions of Class B
};
Class C : access modifier A
{
        //Data members and member functions of Class C
};
Class D: access modifier B, access modifier C
{
        //Data members and member functions of Class D
};
```

**2. What is the role of an Exception Handler in a programming language? Briefly explain important tasks it performs.**

**Ans.**

An Exception Handler is used to manage runtime errors in a program. When an error (exception) occurs, the exception handler intercepts it, allowing the program to handle it gracefully without crashing.

**Tasks it performs:**

- **try :** A block where code that may throw an exception is placed.
- **throw :** Used to signal that an exception has occurred.
- **catch :** A block that handles the exception if it occurs.
- **finally :** An optional block that executes after try and catch, used for cleanup activities.

**Syntax:**

```
try {
   // Code that might throw an exception
} catch (ExceptionType e) {
   // Code to handle the exception
   throw e; // Optional: re-throw the exception
} finally {
   // Code that always runs
}
```

**Example:**

```
try {
   // Code that might throw an exception
   int result = 10 / 0; // This will throw an ArithmeticException
} catch (ArithmeticException e) {
```

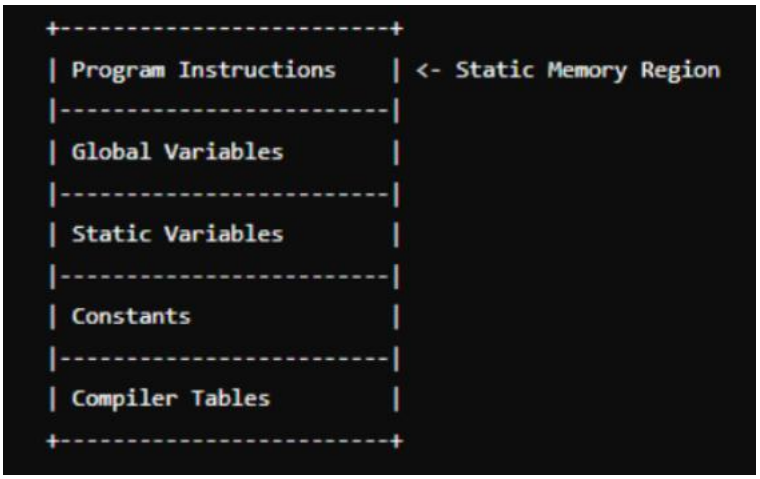
```
    // Code to handle the exception
    System.out.println("An error occurred: " + e.getMessage());
    throw e; // Re-throw the exception if necessary
} finally {
    // Code that will always run, regardless of whether an exception was thrown
    System.out.println("This block is always executed.");
}
```

**3. Explain different storage allocation mechanisms.**

**Ans.**

**There are two primary storage allocation mechanisms:**

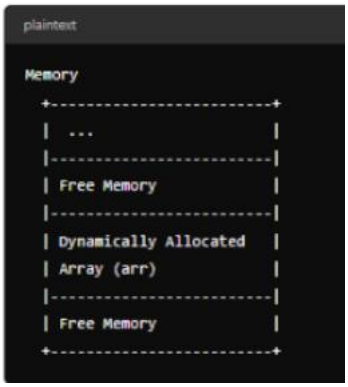
- **Static Allocation :** Memory for variables is allocated at compile time. The size and location of the memory are fixed throughout the program.




**Example:**

```c
#include <stdio.h>

int globalVar = 10; // Global variable

void func() {
    static int staticVar = 5; // Static variable
    printf("Static Variable: %d\n", staticVar);
    staticVar++;
}

int main() {
    func();
    func();
    return 0;
}
```

- **Dynamic Allocation :** Memory is allocated during runtime, allowing for flexible use of memory. Common functions for dynamic allocation in C/C++ are malloc(), calloc(), realloc(), and free().




**Example:**

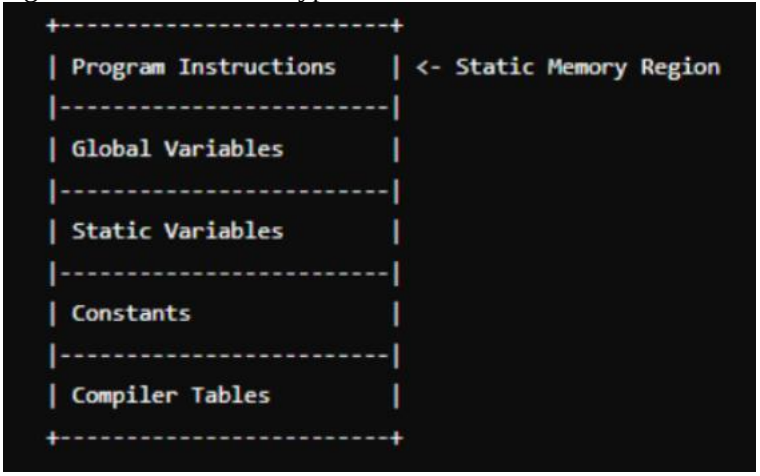
```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int *arr;
    arr = (int *)malloc(5 * sizeof(int)); // Allocate memory for an array
    if (arr == NULL) {
        printf("Memory allocation failed\n");
        return 1;
    }

    for (int i = 0; i < 5; i++) {
        arr[i] = i * 10; // Assign values to the array
    }

    for (int i = 0; i < 5; i++) {
        printf("%d ", arr[i]); // Print the array values
    }

    free(arr); // Free the allocated memory
    return 0;
}
```

**4. Types of bindings in Programming Languages.**

**Ans.**

Binding refers to the association of program elements, such as variables, functions, or objects, with attributes or memory locations.

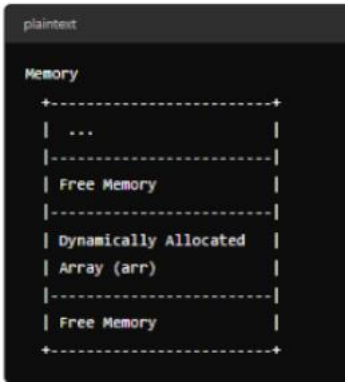
- **Static Binding (Early Binding) :** The association is made at compile time. It's used for overloading and when exact data types or function calls are known at compile time.




**Example:**

```c
#include <stdio.h>

int globalVar = 10; // Global variable

void func() {
    static int staticVar = 5; // Static variable
    printf("Static Variable: %d\n", staticVar);
    staticVar++;
}

int main() {
    func();
    func();
    return 0;
}
```

- **Dynamic Binding (Late Binding) :** The association is made at runtime, typically used in the context of polymorphism where the exact method to be invoked is determined at runtime.




**Example:**

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int *arr;
    arr = (int *)malloc(5 * sizeof(int)); // Allocate memory for an array
    if (arr == NULL) {
        printf("Memory allocation failed\n");
        return 1;
    }

    for (int i = 0; i < 5; i++) {
        arr[i] = i * 10; // Assign values to the array
    }

    for (int i = 0; i < 5; i++) {
        printf("%d ", arr[i]); // Print the array values
    }

    free(arr); // Free the allocated memory
    return 0;
}
```

**5. Compare Static Scoping vs. Dynamic Scoping**

**Ans.**

| Feature | Static Scoping (Lexical Scoping) | Dynamic Scoping |
|---|---|---|
| **Definition** | Scope determined at compile time. | Scope determined at runtime. |
| **Resolution Method** | Based on the structure of the code (lexical context). | Based on the call stack (execution context). |
| **Example Languages** | C, C++, Java, Python | Lisp, Perl (optional) |
| **Variable Access** | Accesses variables in the block where they are declared. | Accesses the most recent variable in the calling context. |
| **Predictability** | More predictable and easier to understand. | Less predictable; behavior can change based on call order. |
| **Performance** | Typically better performance (resolved at compile time). | May incur overhead due to runtime resolution. |
| **Flexibility** | Less flexible in accessing outer scopes. | More flexible in accessing variables across different scopes. |
| **Advantages** | Reduces chances of side effects; clearer scope. | Can simplify certain programming patterns; allows dynamic behavior. |
| **Disadvantages** | Limited access to variables outside the local scope. | Can lead to confusion and difficult debugging. |

**6. How to choose a programming language?**

**Ans.**

**Choosing a programming language depends on several factors:**

- **Purpose :** What is the project requirement? (e.g., web development, system programming, data analysis).
- **Performance :** Some languages like C/C++ are faster and more efficient for system-level programming.
- **Community and Support :** Languages with a large community (like Python) offer more libraries,
- **Learning Curve :** How easy is it to learn? Python is known for its simplicity.
- **Tools and Ecosystem :** The availability of tools and libraries relevant to the project.

**7. Explain the need of different programming paradigms.**

**Ans.**

As technology evolves and becomes more complex, the problems that programmers need to solve also become more complex. Advanced concepts such as machine learning and artificial intelligence involve processing vast amounts of data and making real-time decisions, which require programming approaches that can efficiently manage these tasks. Traditional programming styles, which may have been sufficient for simpler, smaller-scale applications, often fall short when faced with the demands of modern technology.

**Example:**

- **Machine Learning** requires paradigms that support mathematical and statistical computations, and can handle large-scale data processing efficiently.
- **Artificial Intelligence** often involves paradigms that support symbolic reasoning, pattern recognition, and can simulate human-like decision-making processes.
- **Big Data** requires paradigms that can manage, process, and analyze large volumes of data quickly and efficiently.
- **Cloud Computing** involves paradigms that support distributed computing, allowing applications to run seamlessly across multiple servers and environments.

**. What do you mean by Programming Paradigm? Explain with example the difference between declarative and imperative programming paradigm**

**Ans.**

A programming paradigm defines a style or "way" of programming. It is a fundamental approach or methodology that guides the way programs are written and structured. Different paradigms offer different ways to think about and solve programming problems, and each paradigm provides its own unique set of principles, techniques, and tools.

| Aspect | Imperative Programming | Declarative Programming |
|---|---|---|
| **Definition** | Focuses on **how** to perform tasks using explicit commands. | Focuses on **what** the desired outcome is without specifying the steps. |
| **State Changes** | Explicitly changes the program's state through assignments and instructions. | Does not explicitly manage state changes; describes relationships and logic. |
| **Examples of Languages** | C, C++, Java, Python (when using imperative constructs). | SQL, HTML, Lisp, Prolog, Python (when using functional features). |
| **Readability** | Can become complex and verbose with many statements and instructions. | Generally more concise and easier to read, focusing on the outcome rather than the process. |
| **Example of Task** | Calculating the sum of numbers using a loop. | Calculating the sum of numbers using a built-in function. |
| **Example Code** | **Imperative Example in Python:**<br>"python<br>Sum_result = 0<br>For i in range (1,6):<br>Sum_result += i<br>print("Sum:",Sum_result) | **Declarative Example in Python:**<br>**"**python<br>Sum_result = sum(range(1,6))<br>print("Sum:",Sum_result)<br>" |

**9. Explain encapsulation. How does it differ from abstraction?**

**Ans.**

Encapsulation is the process or method to contain the information. Encapsulation is a method to hide the data in a single entity or unit along with a method to protect information from outside world. This method encapsulates the data and function together inside a class which also results in data abstraction.

**Example:**

```
class ClassName {
private:
   // Private data members
   DataType variable1;
   DataType variable2;

public:
   // Public methods (getters and setters)
   ReturnType getVariable1() {
      return variable1;
   }

   void setVariable1(DataType value) {
      variable1 = value;
   }

   // Other public methods
};
```

| Aspect | Encapsulation | Abstraction |
|---|---|---|

| | | |
|---|---|---|
| **Definition** | Bundling of data (variables) and methods that operate on the data into a single unit (class). | Hiding the implementation details and exposing only the essential features of an object. |
| **Focus** | Protecting the internal state of an object and controlling access to its data. | Reducing complexity by providing a simplified model of an object. |
| **Purpose** | Preventing unintended interference and misuse of data. | Allowing users to interact with objects at a high level without needing to understand their complexities. |
| **Implementation** | Achieved using access modifiers (private, protected, public). | Achieved through abstract classes, interfaces, or simply hiding implementation details. |
| **Example** | A class with private variables and public methods to access them. | An abstract class or interface that defines methods without providing implementations. |
| **Type of Mechanism** | A way of organizing data and behavior. | A way of designing software and defining object behavior. |
| **Usage** | Ensures data integrity and security. | Provides a clear model and reduces the cognitive load for users. |

**10. What is the role of an Exception Handler in a programming language? Briefly explain important tasks it performs(explain try,throw,catch in exception hadling).**

**Ans.**

An Exception Handler is used to manage runtime errors in a program. When an error (exception) occurs, the exception handler intercepts it, allowing the program to handle it gracefully without crashing.

**Tasks it performs:**

- **try :** A block where code that may throw an exception is placed.
- **throw :** Used to signal that an exception has occurred.
- **catch :** A block that handles the exception if it occurs.
- **finally :** An optional block that executes after try and catch, used for cleanup activities.

**11. Explain Encapsulation and Abstraction with suitable examples from C++ or Java**

**Ans.**

**Encapsulation:**

Encapsulation is the process or method to contain the information. Encapsulation is a method to hide the data in a single entity or unit along with a method to protect information from outside world. This method encapsulates the data and function together inside a class which also results in data abstraction.

**Abstraction:**

Abstraction is the method of getting information where the information needed will be taken in such a simplest way that solely the required components are extracted, and also the ones that are considered less significant are unnoticed. The concept of abstraction only shows necessary information to the users. It reduces the complexity of the program by hiding the implementation complexities of programs.

**Encapsulation Example (Java):**

```java
class Student {
    private String name; // Private variable
    public String getName() { // Public method to access private variable
        return name;
    }
    public void setName(String newName) {
        name = newName;
    }
}
```

**Abstraction Example (Java):**

```java
abstract class Animal {
    abstract void sound(); // Abstract method
}
class Dog extends Animal {
    void sound() {
        System.out.println("Bark");
    }
}
```

**12. What are different programming paradigms? Give one example of each.**

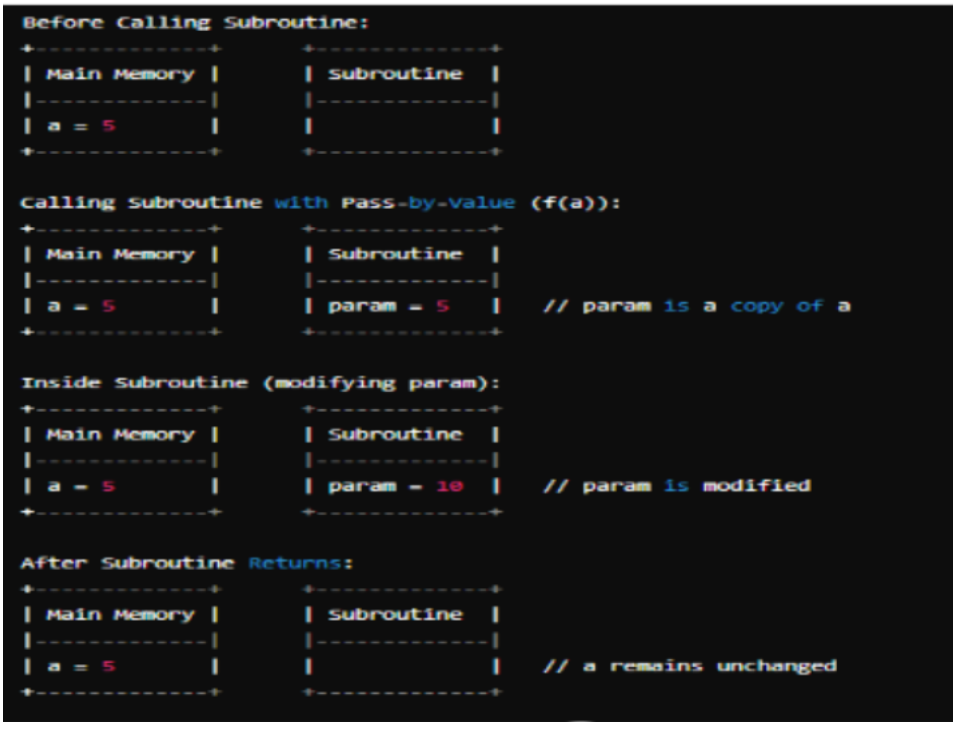
**Ans.**

- **Procedural Programming :** C
- **Object-Oriented Programming (OOP) :** Java
- **Functional Programming :** Haskell
- **Logical Programming :** Prolog
- **Event-Driven Programming :** JavaScript

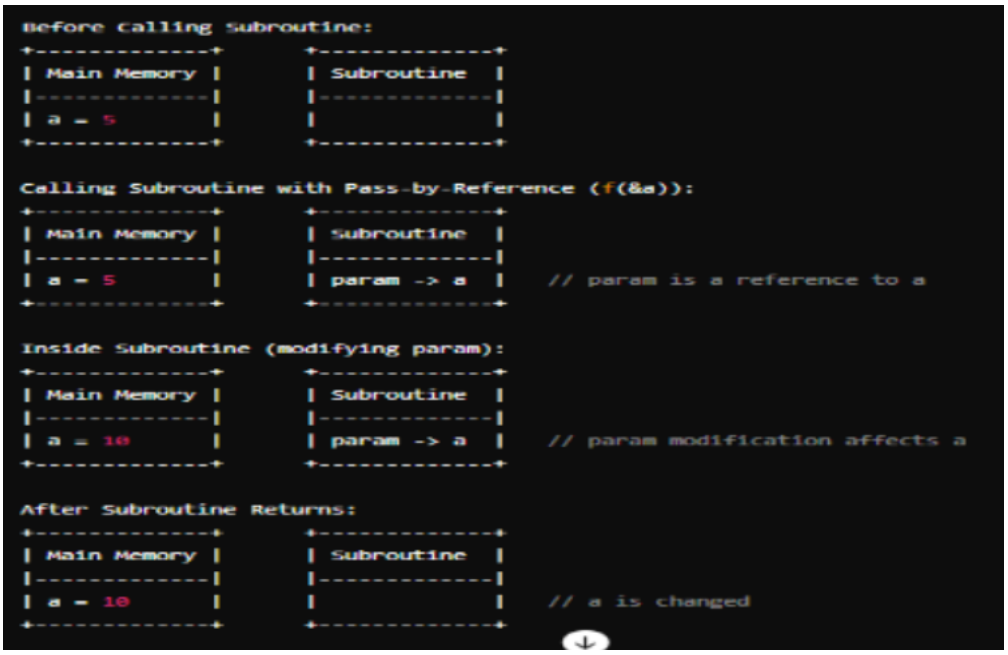
**13. Discuss parameter passing in subroutine or explain Call by value vs Call by reference with example code in C or C++.**

**Ans.**

**Call by Value :** The actual value is passed to the function. Changes made to the parameter inside the function do not affect the argument.



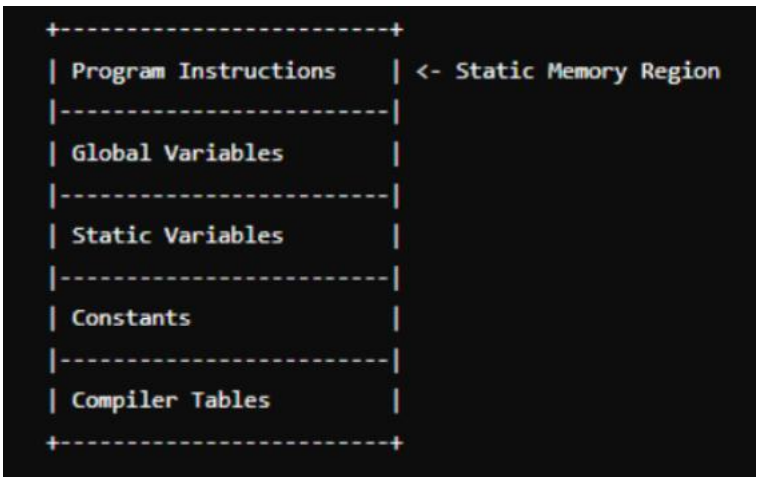
**Call by Reference :** The address of the variable is passed to the function. Changes made inside the function affect the original variable.




**14. Explain static allocation method with example.**

**Ans.**

- **Static Allocation :** Memory for variables is allocated at compile time. The size and location of the memory are fixed throughout the program.



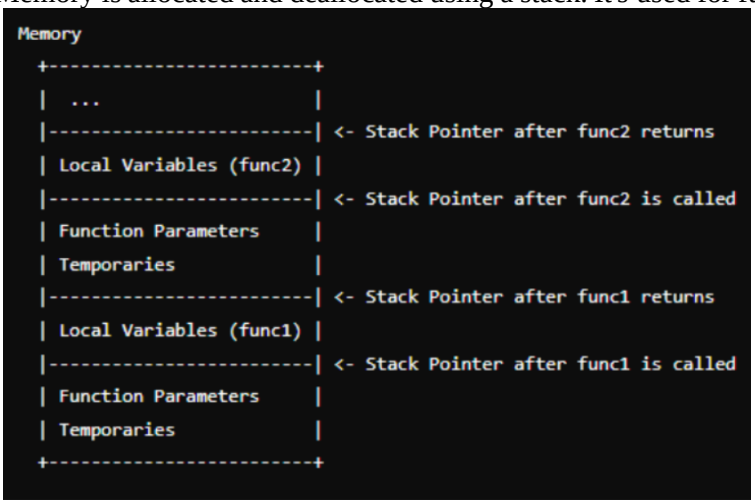

**Example:**

```c
#include <stdio.h>

int globalVar = 10; // Global variable

void func() {
    static int staticVar = 5; // Static variable
    printf("Static Variable: %d\n", staticVar);
    staticVar++;
}

int main() {
    func();
    func();
    return 0;
}
```

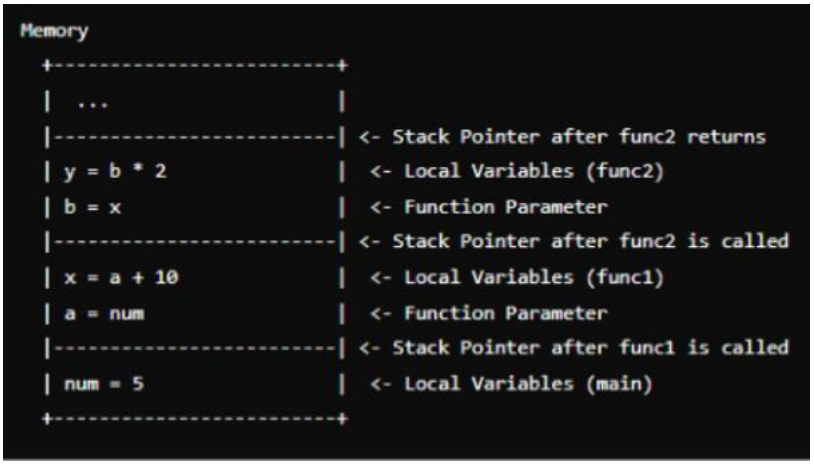
**15. Explain stack allocation method with example.**

**Ans.**

**Stack Allocation :** Memory is allocated and deallocated using a stack. It's used for function calls.




**Example:**

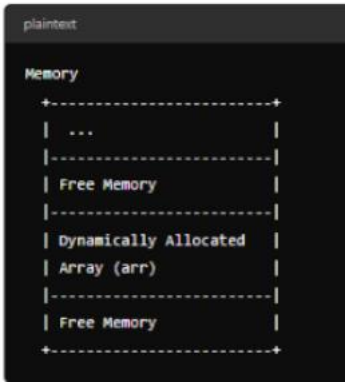



**16. Explain heap allocation method with example.**

**Ans.**

**Heap Allocation :** Memory is allocated at runtime using dynamic memory allocation functions like

malloc() and free() in C.




**Example:**

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int *arr;
    arr = (int *)malloc(5 * sizeof(int)); // Allocate memory for an array
    if (arr == NULL) {
        printf("Memory allocation failed\n");
        return 1;
    }

    for (int i = 0; i < 5; i++) {
        arr[i] = i * 10; // Assign values to the array
    }

    for (int i = 0; i < 5; i++) {
        printf("%d ", arr[i]); // Print the array values
    }

    free(arr); // Free the allocated memory
    return 0;
}
```

**17. What are constructor and destructor? Explain with example.**
**Ans.**
**Constructor :** A special method called automatically when an object is created to initialize the object.
**Destructor :** A special method called automatically when an object is destroyed to clean up resources.

**Example in C++:**

```cpp
class MyClass {
public:
    MyClass() { // Constructor
        cout << "Object created";
    }
    ~MyClass() { // Destructor
        cout << "Object destroyed";
    }
};
```

**18. Explain different types of constructor with example.**
**Ans.**
**Default Constructor :** Takes no parameters.

**Parameterized Constructor :** Takes parameters to initialize objects with specific values.
**Copy Constructor :** Used to create a new object as a copy of an existing object.

```
Example in C++:
cpp
class MyClass {
public:
   MyClass() { // Default Constructor
      cout << "Default constructor";
   }
   MyClass(int x) { // Parameterized Constructor
      cout << "Parameterized constructor";
   }
   MyClass(const MyClass &obj) { // Copy Constructor
      cout << "Copy constructor";
   }
};
```

**19. Explain class,object with example.**

**Ans.**

A Class is a blueprint for creating objects (instances). It defines properties and behaviors (attributes and methods).

An Object is an instance of a class.

```
Example in Java:
class Car { // Class
   String color; // Attribute
   void start() { // Method
      System.out.println("Car started");
   }
}

public class Main {
   public static void main(String[] args) {
      Car myCar = new Car(); // Object
      myCar.color = "Red";
      myCar.start();
   }
```

# PROLOG

**1. Explain the concept of Unification and Resolution with an example.**

**Ans.**

## Resolution

- In order to derive new statements, a logic programming system combines existing statements, canceling like terms, through a process known as resolution.
- If we Resolution know that A and B imply C, for example, and that C implies D, we can deduce that A and B imply D:

$$\frac{C \leftarrow A, B \qquad D \leftarrow C}{D \leftarrow A, B}$$

- During resolution, free variables may acquire values through unification with expressions in matching terms, much as variables acquire types in ML

$$\frac{\text{flowery}(X) \leftarrow \text{rainy}(X) \qquad \text{rainy}(\text{Rochester})}{\text{flowery}(\text{Rochester})}$$

?-X = Y → True (if they unify); False (otherwise)

built in

X = Y
Y = 10
X= 10

Rules:-

i) X and Y are permitted to be uninstantiated variables.
ii) Integers and atoms are always equal to themselves.
iii) 2 structures are equal only if :-
   a) Same functions.
   b) No. of components.
   c) corresponding components are equal.

Eg. a(b,c(d,e))= a(X, c(Y, e))
X = b
Y = d

```
?- police=police.
true.

?- police=pen.
false.

?- 686=686.
true.

?- 686=2.
false.

?- a(b,c(d,e))=a(X,c(Y,e)).
X = b,
Y = d.

?- a(b,c(d,e))=a(X,u(Y,e)).
false.

?-
```

**2. What is Backtracking in Prolog? how backtracking can be prevented. Give an example.**

**Ans.**

**Backtracking:**

The process of finding alternative ways of satisfying a goal by entering a semicolon at the system prompt is known as backtracking, or more precisely 'forcing the Prolog system to backtrack

**Example:**

- The cut, in Prolog, is a goal, written as !, which always succeeds, but cannot be backtracked past.
- It is used to prevent unwanted backtracking, for example, to prevent extra solutions being found by Prolog.

- Example: Suppose we have the following facts:
- teaches(dr_fred, history). studies(alice, english).
- teaches(dr_fred, english). studies(angus, english).
- teaches(dr_fred, drama). studies(amelia, drama).
- teaches(dr_fiona, physics). studies(alex, physics).

```
?- !, teaches(dr_fred, Course), studies(Student, Course).

   Course = english
   Student = alice ;

   Course = english
   Student = angus ;

   Course = drama
   Student = amelia ;

   false.
   ?-
```

**3.How cuts can be used in prolog to prevent backtracking?**
**Ans.**

## Cuts In Prolog


- The cut, in Prolog, is a goal, written as !, which always succeeds, but cannot be backtracked past.
- It is used to prevent unwanted backtracking, for example, to prevent extra solutions being found by Prolog.

- Example: Suppose we have the following facts:
- teaches(dr_fred, history). studies(alice, english).
- teaches(dr_fred, english). studies(angus, english).
- teaches(dr_fred, drama). studies(amelia, drama).
- teaches(dr_fiona, physics). studies(alex, physics).

```
?- !, teaches(dr_fred, Course), studies(Student, Course).

   Course = english
   Student = alice ;

   Course = english
   Student = angus ;

   Course = drama
   Student = amelia ;

   false.
   ?-
```


**4.Define Facts, rules and knowledge based in prolog with example.**

**Ans.**
**Facts:**

a. Facts are statements about what is true about a problem, instead of instructions how to accomplish the solution.
b. The Prolog system uses the facts to work out how to accomplish the solution by searching through the space of possible solutions.
c. It is defined by an identifier followed by an n-tuple of constants.
d. A relation identifier is referred to as a predicate
e. When a tuple of values is in a relation we say the tuple satisfies the predicate.

**Example:**

| Predicate | Interpretation |
|---|---|
| valuable(gold) | Gold is valuable. |
| owns(john,gold) | John owns gold. |
| father(john,mary) | John is the father of Mary |
| gives (john,book,mary) | John gives the book to Mary |

**Rules:**

f. Specifies under what conditions a tuple of values satisfies a predicate.
g. The basic building block of a rule is called an atom
h. Atom :- Atom1, ..., Atomn
i. If each of Atom1,...,Atomn is true, then Atom is also true.
j. In almost all logic languages, axioms are written in a standard form known as a Horn clauses
k. A Horn clause consists of a head, or consequent term H, and a body consisting of terms Bi:
H ← B1, B2,..., Bn
l. The semantics of this statement are that when the Bi are all true, we can deduce that H is true as well.
m. When reading aloud, we say "H, if B1, B2, . . . , and Bn."
n. Horn clauses can be used to capture most, but not all, logical statements.