# **Team-3 JOI Programming Language**

**ARUP BISWAS (CS21B004)**

**SRINIVAS JATOTHU (CS21B023)**

**NIRAJ KUMAR (CS21B038)**

# **Documentation**

## **Introduction**

The JOI language is a new, modern programming language similar to C++, designed for efficiency and simplicity in handling complex programs. This document outlines the key syntax, grammar, and language features to help the compiler team understand and implement the JOI compiler.

### **Language Specification Overview: JOI**

#### **1. Name and Version**

* **Language Name**: JOI (wip)
* **Version**: 1.0

#### **2. Syntax**

* **Basic Structure**: Clean syntax resembling C++, using {} for blocks and ; for statements.
* **Comments**: we will use \*--\*

#### **3. Data Types**

* **Primitive Types**: int, float, string, bool.
  + Sizes: 4, 4, <str\_len>\*1, 1, 1 byte(s)
* **Composite Types**: array, map, pointer, struct.
* **Collections**: queue, stack.

#### **4. Control Structures**

* **Conditional Statements**: if, else if, else, switch. (nested?)
* **Loops**: for, while.

#### **5. Functions**

* **Definition**: function name(parameters) { /\* body \*/ }.
* **First-Class Support**: Functions can be passed as arguments, returned, and assigned.

#### **6. Error Handling**

* **Try/Catch**: try { /\* code \*/ } catch (error) { /\* handle error \*/ }.

#### **7. Concurrency**

* **Asynchronous Functions**: Use async for asynchronous functions, with await for promises.

#### **8. Standard Library**

* **Core Modules**: Modules like
* **Example Functions**:

#### **9. Input/Output**

* **Print Statement**: Use print("message") for console output.
* **Input Function**: Use input("prompt") to read user input.

### **Additional Data Structures**

#### **Pointers**

* **Definition**: Variables that hold memory addresses for efficient manipulation and dynamic allocation.

#### **Stacks**

* **Definition**: Collection following Last-In-First-Out (LIFO).
  + push(value): Add an element to the top.
  + pop(): Remove and return the top element.

#### **Queues**

* **Definition**: Collection following First-In-First-Out (FIFO).
  + enqueue(value): Add an element to the end.
  + dequeue(): Remove and return the front element.

### **Object-Oriented Programming Concepts**

#### **11. Classes and Objects**

* **Class Definition**: Use the syntax class ClassName { /\* properties and methods \*/ } to define a class.
* **Object Creation**: Instantiate objects with new ClassName().

#### **12. Inheritance**

* **Single Inheritance**: Allow classes to inherit properties and methods from a single parent class using the syntax class ChildClass : ParentClass { /\* body \*/ }.
* **Method Overriding**: Support overriding parent class methods in derived classes.

#### **13. Encapsulation**

* **Access Modifiers**: Use public, private, and protected to control access to class members.
* **Getters and Setters**: Implement methods to access and modify private properties.

#### **14. Polymorphism**

* **Method Overloading**: Allow multiple methods with the same name but different parameters.
* **Dynamic Polymorphism**: Enable method overriding in derived classes.

#### **15. Interfaces**

* **Interface Definition**: Define interfaces with interface InterfaceName { /\* method signatures \*/ }.
* **Implementation**: Use implements keyword in class definitions to adhere to an interface.

### **Development Steps for OOP**

1. **Design Class Syntax**:
   * Define grammar rules for class, object, inheritance, and access modifiers.
2. **Implement Class System**:
   * Create a framework for defining classes, methods, and properties.
3. **Support Inheritance**:
   * Allow classes to extend other classes and manage method overriding.
4. **Add Access Modifiers**:
   * Implement public, private, and protected to control member accessibility.
5. **Implement Polymorphism**:
   * Create mechanisms for method overloading and overriding.
6. **Create Interfaces**:
   * Develop the syntax and structure for defining and implementing interfaces.

Language Syntax

**Overview:** In JOI, every program is structured around a main function, which serves as the entry point for execution. The main function is defined using the int joi() syntax. This function can be followed by additional functions and declarations as needed.

Syntax:

#include <iostream>

using namespace std;

int joi() {

return 0;

}



Example:



#include <iostream>

using namespace std;

int joi() {

cout << "Please enter your name: "; ## Prompt user

cin >> name; ## Read input from user using cin

cout << "Hello, " << name << "!" << endl; ## Greet the user

return 0; ## Indicate successful execution

}



The below code is for suing input and output keywords for taking i/p and o/p

##io.joi - A simple input/output library for JOI

string input() {

string value = ""; // Initialize an empty string

char ch;

// Simulate reading from standard input until Enter is pressed

while (true) {

ch = getchar(); // Read a character

// Break if Enter is pressed

if (ch == '\n') {

break;

}

value += ch; // Append character to the string

}

return value; // Return the collected input

}

// Function to display output

void output(string message) {

// Simulate writing to standard output

// Loop through each character in the message

for (int i = 0; i < message.length(); i++) {

putchar(message[i]); // Write each character to output

}

}



// Import the IO library

import "io.joi";

int joi() {

output("Please enter your name: "); // Prompt user

string name = input(); // Get user input

output("Hello, " + name + "!"); // Greet the user

return 0; // Indicate successful execution

}



# Data Types

## Data Type Syntax in JOI

## Integer

int variableName = value;

## Floating Point

float variableName = value;

## Character

char variableName = 'value';

## Boolean

bool variableName = true/false;



## main.joi - Example Program Demonstrating Data Types

#include <iostream>

using namespace std;

int joi() {

int age = 25; ## Integer

float height = 5.9; ## Floating Point

char grade = 'A'; ## Character

bool isStudent = true; ## Boolean

char name[100]; ## String (Character Array)

cout << "Please enter your name: ";

cin >> name; ## Read input for name

cout << "Hello, " << name << "!" << endl;

cout << "Age: " << age << endl;

cout << "Height: " << height << " feet" << endl;

cout << "Grade: " << grade << endl;

cout << "Is Student: " << (isStudent ? "Yes" : "No") << endl;

return 0; ## Indicate successful execution

}



### Declaring Multiple Variables with One Value

dataType var1, var2, var3 = value;



## main.joi - Example Program for Declaring Multiple Variables

#include <iostream>

using namespace std;

int joi() {

// Declare multiple integer variables with one value

int a, b, c = 10; ## All three variables will be initialized to 10

// Output the values

cout << "Value of a: " << a << endl; // a will be uninitialized

cout << "Value of b: " << b << endl; // b will be uninitialized

cout << "Value of c: " << c << endl; // c will be 10

// To initialize all to the same value, do it explicitly

a = b = c = 10; ## Assign the same value to a and b as well

// Output the updated values

cout << "Updated value of a: " << a << endl;

cout << "Updated value of b: " << b << endl;

cout << "Updated value of c: " << c << endl;

return 0; ## Indicate successful execution

}



### Declaring Constants in JOI

constant dataType constantName = value;



#### One example to test the above syntax

## main.joi - Example Program for Calculating Total Cost

### This program calculates the total cost of fruits based on user input.

#include <iostream>

using namespace std;

int joi() {

constant float price\_apple = 1.50; ## Price per apple

constant float price\_banana = 0.75; ## Price per banana

constant float price\_orange = 1.25; ## Price per orange

## Variables for quantities

int quantity\_apples, quantity\_bananas, quantity\_oranges;

## Prompt user for quantities

cout << "Enter the quantity of apples: ";

cin >> quantity\_apples;

cout << "Enter the quantity of bananas: ";

cin >> quantity\_bananas;

cout << "Enter the quantity of oranges: ";

cin >> quantity\_oranges;

## Calculate total cost

float total\_cost = (price\_apple \* quantity\_apples) +

(price\_banana \* quantity\_bananas) +

(price\_orange \* quantity\_oranges);

## Output the total cost

cout << "Total cost: $" << total\_cost << endl;

return 0; ## Indicate successful execution

}







# Arithmetic operations

## Addition

result = operand1 + operand2;

## Subtraction

result = operand1 - operand2;

## Multiplication

result = operand1 \* operand2;

## Division

result = operand1 / operand2;

## Modulus

result = operand1 % operand2; ## Remainder of division



#### Example

## main.joi - Example Program for Arithmetic Operations

#include <iostream>

using namespace std;

int joi() {

int a, b; ## Variables for user input

int sum, difference, product, quotient, remainder; ## Variables for results

## Prompt user for input

cout << "Enter two integers: ";

cin >> a >> b;

## Perform arithmetic operations

sum = a + b; ## Addition

difference = a - b; ## Subtraction

product = a \* b; ## Multiplication

quotient = a / b; ## Division

remainder = a % b; ## Modulus

## Output the results

cout << "Sum: " << sum << endl;

cout << "Difference: " << difference << endl;

cout << "Product: " << product << endl;

cout << "Quotient: " << quotient << endl;

cout << "Remainder: " << remainder << endl;

return 0; ## Indicate successful execution

}



### Assignment Operators

## Simple Assignment

variable = value;

## Add and Assign

variable += value; ## Equivalent to variable = variable + value;

## Subtract and Assign

variable -= value; ## Equivalent to variable = variable - value;

## Multiply and Assign

variable \*= value; ## Equivalent to variable = variable \* value;

## Divide and Assign

variable /= value; ## Equivalent to variable = variable / value;

## Modulus and Assign

variable %= value; ## Equivalent to variable = variable % value;



## main.joi - Example Program for Assignment Operators

#include <iostream>

using namespace std;

int joi() {

int value = 10; ## Initialize a variable

int result; ## Variable to store results

## Using assignment operators

result = value; ## Simple assignment

cout << "Initial result: " << result << endl;

result += 5; ## Add and assign

cout << "After adding 5: " << result << endl;

result -= 3; ## Subtract and assign

cout << "After subtracting 3: " << result << endl;

result \*= 2; ## Multiply and assign

cout << "After multiplying by 2: " << result << endl;

result /= 4; ## Divide and assign

cout << "After dividing by 4: " << result << endl;

result %= 3; ## Modulus and assign

cout << "After modulus with 3: " << result << endl;

return 0; ## Indicate successful execution

}



## Equal to

result = (operand1 == operand2); ## Returns true if equal

## Not equal to

result = (operand1 != operand2); ## Returns true if not equal

## Greater than

result = (operand1 > operand2); ## Returns true if operand1 is greater

## Less than

result = (operand1 < operand2); ## Returns true if operand1 is less

## Greater than or equal to

result = (operand1 >= operand2); ## Returns true if operand1 is greater or equal

## Less than or equal to

result = (operand1 <= operand2); ## Returns true if operand1 is less or equal



### Comparison Operators in JOI

## Equal to

result = (operand1 == operand2); ## Returns true if equal

## Not equal to

result = (operand1 != operand2); ## Returns true if not equal

## Greater than

result = (operand1 > operand2); ## Returns true if operand1 is greater

## Less than

result = (operand1 < operand2); ## Returns true if operand1 is less

## Greater than or equal to

result = (operand1 >= operand2); ## Returns true if operand1 is greater or equal

## Less than or equal to

result = (operand1 <= operand2); ## Returns true if operand1 is less or equal



## main.joi - Example Program for Comparison Operators

#include <iostream>

using namespace std;

int joi() {

int a, b; ## Variables for user input

bool isEqual, isNotEqual, isGreater, isLess, isGreaterOrEqual, isLessOrEqual;

## Prompt user for input

cout << "Enter two integers: ";

cin >> a >> b;

## Using comparison operators

isEqual = (a == b); ## Equal to

isNotEqual = (a != b); ## Not equal to

isGreater = (a > b); ## Greater than

isLess = (a < b); ## Less than

isGreaterOrEqual = (a >= b); ## Greater than or equal to

isLessOrEqual = (a <= b); ## Less than or equal to

## Output the results

cout << "Is Equal: " << (isEqual ? "True" : "False") << endl;

cout << "Is Not Equal: " << (isNotEqual ? "True" : "False") << endl;

cout << "Is Greater: " << (isGreater ? "True" : "False") << endl;

cout << "Is Less: " << (isLess ? "True" : "False") << endl;

cout << "Is Greater or Equal: " << (isGreaterOrEqual ? "True" : "False") << endl;

cout << "Is Less or Equal: " << (isLessOrEqual ? "True" : "False") << endl;

return 0; ## Indicate successful execution

}



### Logical Operators

## Logical AND

result = (condition1 && condition2); ## Returns true if both conditions are true

## Logical OR

result = (condition1 || condition2); ## Returns true if at least one condition is true

## Logical NOT

result = !(condition); ## Returns true if the condition is false



## main.joi - Example Program for Logical Operators

#include <iostream>

using namespace std;

int joi() {

int a, b; ## Variables for user input

bool isTrue, isFalse;

## Prompt user for input

cout << "Enter two integers: ";

cin >> a >> b;

## Using logical operators

isTrue = (a > 0 && b > 0); ## Logical AND

isFalse = (a < 0 || b < 0); ## Logical OR

bool notTrue = !(a == b); ## Logical NOT

## Output the results

cout << "Both numbers positive: " << (isTrue ? "True" : "False") << endl;

cout << "At least one number negative: " << (isFalse ? "True" : "False") << endl;

cout << "Numbers are not equal: " << (notTrue ? "True" : "False") << endl;

return 0; ## Indicate successful execution

}



# 

## 

# Strings

str variableName; ## Declare a string variable

### Concatenation

str result = string1 + string2; ## Concatenates two strings



### Accessing Characters

char character = str [index]; ## Access character at the specified index



### Changing String Characters

str [index] = newCharacter; ## Change character at the specified index



## main.joi - Example Program for String Operations

#include <iostream>

using namespace std;

int joi() {

str firstName, lastName; ## Variables for user input

str fullName; ## Variable for concatenated result

## Prompt user for input

cout << "Enter your first name: ";

cin >> firstName;

cout << "Enter your last name: ";

cin >> lastName;

## Concatenate strings

fullName = firstName + " " + lastName; ## Concatenating first and last name

cout << "Full Name: " << fullName << endl;

## Accessing characters

char firstChar = fullName[0]; ## Get the first character of the full name

cout << "First character of Full Name: " << firstChar << endl;

## Changing a character

fullName[0] = 'X'; ## Change first character to 'X'

cout << "Modified Full Name: " << fullName << endl;

return 0; ## Indicate successful execution

}



# Booleans

### Declaration

bool variableName; ## Declare a boolean variable



### Assigning Values

variableName = true; ## Assigning true

variableName = false; ## Assigning false



### Using Boolean Expressions

bool result = (condition1 && condition2); ## Logical AND

bool result = (condition1 || condition2); ## Logical OR

bool result = !(condition); ## Logical NOT



## main.joi - Example Program for Booleans

#include <iostream>

using namespace std;

int joi() {

int a, b; ## Variables for user input

bool isEqual, isGreater; ## Boolean variables for conditions

## Prompt user for input

cout << "Enter two integers: ";

cin >> a >> b;

## Using boolean expressions

isEqual = (a == b); ## Check if both numbers are equal

isGreater = (a > b); ## Check if the first number is greater

## Output the results

cout << "Are the numbers equal? " << (isEqual ? "True" : "False") << endl;

cout << "Is the first number greater than the second? " << (isGreater ? "True" : "False") << endl;

return 0; ## Indicate successful execution

}



# IF ELSE

### If Statement

if condition: {

## Code to execute if condition is true

}

### Else Statement

else: {

## Code to execute if condition is false

}



### Else If Statement

else if anotherCondition: {

## Code to execute if anotherCondition is true

}



## main.joi - Example Program for Conditional Statements with Modified Syntax

#include <iostream>

using namespace std;

int joi() {

int score; ## Variable for user input

## Prompt user for input

cout << "Enter your score (0-100): ";

cin >> score;

## Conditional statements with modified syntax

if score >= 90: {

cout << "Grade: A" << endl; ## If score is 90 or above

} else if score >= 80: {

cout << "Grade: B" << endl; ## If score is 80 or above

} else if score >= 70: {

cout << "Grade: C" << endl; ## If score is 70 or above

} else if score >= 60: {

cout << "Grade: D" << endl; ## If score is 60 or above

} else: {

cout << "Grade: F" << endl; ## If score is below 60

}

return 0; ## Indicate successful execution

}



# Switch Case

switch expression: {

case value1:

## Code to execute if expression equals value1

break;

case value2:

## Code to execute if expression equals value2

break;

default:

## Code to execute if expression doesn't match any case

}



## main.joi - Example Program for Switch Statement with Modified Syntax

#include <iostream>

using namespace std;

int joi() {

int day; ## Variable for user input

str dayName; ## Variable to store the name of the day

## Prompt user for input

cout << "Enter a number (1-7) for the day of the week: ";

cin >> day;

## Switch statement to determine the day name

switch day: {

case 1:

dayName = "Monday";

break;

case 2:

dayName = "Tuesday";

break;

case 3:

dayName = "Wednesday";

break;

case 4:

dayName = "Thursday";

break;

case 5:

dayName = "Friday";

break;

case 6:

dayName = "Saturday";

break;

case 7:

dayName = "Sunday";

break;

default:

dayName = "Invalid day! Please enter a number between 1 and 7.";

}

## Output the result

cout << "Day: " << dayName << endl;

return 0; ## Indicate successful execution

}



# While Loop

while condition: {

## Code to execute as long as the condition is true

}



## main.joi - Example Program for While Loop with Modified Syntax

#include <iostream>

using namespace std;

int joi() {

int count = 1; ## Initialize the counter variable

cout << "Counting from 1 to 5:" << endl;

## While loop to count from 1 to 5

while count <= 5: {

cout << count << endl; ## Output the current count

count++; ## Increment the counter

}

return 0; ## Indicate successful execution

}



# Do While

do {

## Code to execute

} while condition: {

## Code to execute as long as the condition is true

}



## main.joi - Example Program for Do/While Loop with Modified Syntax

#include <iostream>

using namespace std;

int joi() {

int count = 1; ## Initialize the counter variable

cout << "Counting from 1 to 5:" << endl;

## Do/While loop to count from 1 to 5

do {

cout << count << endl; ## Output the current count

count++; ## Increment the counter

} while count <= 5: { ## Condition to continue looping

## No additional code needed here

}

return 0; ## Indicate successful execution

}



# For Loop

for initialization; condition; increment: {

## Code to execute

}



## main.joi - Example Program for For Loop with Modified Syntax

#include <iostream>

using namespace std;

int joi() {

cout << "Counting from 1 to 5:" << endl;

## For loop to count from 1 to 5

for int count = 1; count <= 5;: count++ : {

cout << count << endl; ## Output the current count

}

return 0; ## Indicate successful execution

}



## Nested For Loops

for initialization1; condition1; increment1: {

for initialization2; condition2; increment2: {

## Code to execute for the inner loop

}

}



## main.joi - Example Program for Nested For Loops with Modified Syntax

#include <iostream>

using namespace std;

int joi() {

cout << "Multiplication Table (1 to 5):" << endl;

## Outer loop for the first number

for int i = 1; i <= 5; i++ : {

## Inner loop for the second number

for int j = 1; j <= 5; j++: {

cout << i \* j << "\t"; ## Output the product

}

cout << endl; ## New line after each row

}

return 0; ## Indicate successful execution

}



# Break and Constant

break; ## Exits the loop immediately



continue; ## Skips the current iteration and moves to the next



## main.joi - Example Program for Break and Continue with Modified Syntax

#include <iostream>

using namespace std;

int joi() {

cout << "Counting from 1 to 10 (skipping 5):" << endl;

## For loop with break and continue

for int count = 1; count <= 10; count++: {

if count == 5: {

continue; ## Skip the rest of the loop when count is 5

}

if count == 8: {

break; ## Exit the loop when count is 8

}

cout << count << endl; ## Output the current count

}

return 0; ## Indicate successful execution

}



# Arrays

dataType arrayName[arraySize]; ## Declaration of an array



### Access the Elements of an Array

arrayName[index]; ## Accessing an element



## Accessing Elements of an Array

#include <iostream>

using namespace std;

int joi() {

int numbers[5] = {1, 2, 3, 4, 5}; ## Declaration and initialization

## Access and print the third element (index 2)

cout << "The third element is: " << numbers[2] << endl; ## Output: 3

return 0;

}



### Change an Array Element

arrayName[index] = newValue; ## Changing an element



## Changing an Array Element

#include <iostream>

using namespace std;

int joi() {

int numbers[5] = {1, 2, 3, 4, 5}; ## Declaration and initialization

## Change the value of the second element (index 1)

numbers[1] = 10; ## Changing value from 2 to 10

## Print the modified array

cout << "Modified array elements:" << endl;

for int i = 0; i < 5; i++: {

cout << numbers[i] << endl; ## Output: 1, 10, 3, 4, 5

}

return 0;

}



### Loop Through an Array

for int i = 0; i < arraySize; i++: {

## Access arrayName[i]

}



## Looping Through an Array

#include <iostream>

using namespace std;

int joi() {

int numbers[5] = {1, 2, 3, 4, 5}; ## Declaration and initialization

## Loop through the array and print each element

cout << "Array elements:" << endl;

for int i = 0; i < 5; i++: {

cout << numbers[i] << endl; ## Output: 1, 2, 3, 4, 5

}

return 0;

}



# Multi-Dimensional Arrays:

dataType arrayName[rows][columns]; ## Declaration of a two-dimensional array

### Accessing Elements of a Multi-Dimensional Array

arrayName[rowIndex][columnIndex]; ## Accessing an element

### Changing an Element in a Multi-Dimensional Array

arrayName[rowIndex][columnIndex] = newValue; ## Changing an element

### Looping Through a Multi-Dimensional Array

for int i = 0; i < rows; i++: {

for int j = 0; j < columns; j++: {

## Access arrayName[i][j]

}

}



## main.joi - Example Program for Multi-Dimensional Arrays

#include <iostream>

using namespace std;

int joi() {

int matrix[3][3]; ## Declaration of a 3x3 array

## Initialize the multi-dimensional array

for int i = 0; i < 3; i++: {

for int j = 0; j < 3; j++: {

matrix[i][j] = (i + 1) \* (j + 1); ## Assign values

}

}

## Print the multi-dimensional array

cout << "Matrix elements:" << endl;

for int i = 0; i < 3; i++: {

for int j = 0; j < 3; j++: {

cout << matrix[i][j] << "\t"; ## Output each element

}

cout << endl; ## New line after each row

}

## Change an element

matrix[1][1] = 99; ## Changing the value at row 1, column 1

## Print the modified matrix

cout << "Modified Matrix:" << endl;

for int i = 0; i < 3; i++: {

for int j = 0; j < 3; j++: {

cout << matrix[i][j] << "\t"; ## Output each element

}

cout << endl; ## New line after each row

}

return 0; ## Indicate successful execution

}



# Structure

struct StructureName:

dataType member1; ## First member

dataType member2; ## Second member

:



### Creating Structure Variables

StructureName variableName; ## Declaration of a structure variable



## Creating Structure Variables in JOI

#include <iostream>

using namespace std;

## Define a structure to hold student information

struct Student:

int id; ## Student ID

str name; ## Student Name

float grade; ## Student Grade

:

int joi() {

## Create structure variables

Student student1; ## First student

Student student2; ## Second student

## Initialize the first student

student1.id = 101;

student1.name = "Alice";

student1.grade = 95.5;

## Initialize the second student

student2.id = 102;

student2.name = "Bob";

student2.grade = 89.0;

return 0; ## Indicate successful execution

}



### Accessing Structure Members

variableName.member; ## Accessing a member



## Accessing Structure Members in JOI

#include <iostream>

using namespace std;

## Define a structure to hold student information

struct Student:

int id; ## Student ID

str name; ## Student Name

float grade; ## Student Grade

:

int joi() {

## Create a structure variable

Student student1;

## Initialize the structure members

student1.id = 101;

student1.name = "Alice";

student1.grade = 95.5;

## Access and output the structure members

cout << "Student ID: " << student1.id << endl; ## Output: 101

cout << "Student Name: " << student1.name << endl; ## Output: Alice

cout << "Student Grade: " << student1.grade << endl; ## Output: 95.5

return 0; ## Indicate successful execution

}



# ENUM

enum EnumName:

constant1, ## First constant

constant2, ## Second constant

constant3, ## Third constant

// Add more constants as needed

:



### Creating Enum Variables

Syntax

EnumName variableName; ## Declaration of an enum variable



## main.joi - Example Program for Enums

#include <iostream>

using namespace std;

## Define an enum for days of the week

enum Day:

Sunday,

Monday,

Tuesday,

Wednesday,

Thursday,

Friday,

Saturday

:

int joi() {

## Create an enum variable

Day today;

## Assign a value to the enum variable

today = Wednesday;

## Output the value of the enum variable

cout << "Today is: " << today << endl; ## Output: 3 (Wednesday)

return 0; ## Indicate successful execution

}



# Reference

dataType& referenceName = originalVariable; ## Declaration of a reference



## main.joi - Example Program for References

#include <iostream>

using namespace std;

int joi() {

int number = 42; ## Original variable

int& refNumber = number; ## Reference to the original variable

## Output the original and reference values

cout << "Original number: " << number << endl; ## Output: 42

cout << "Reference number: " << refNumber << endl; ## Output: 42

## Modify the original variable through the reference

refNumber = 100;

## Output the modified original and reference values

cout << "Modified original number: " << number << endl; ## Output: 100

cout << "Modified reference number: " << refNumber << endl; ## Output: 100

return 0; ## Indicate successful execution

}



# Memory Addresses

# Pointers

### Syntax for Declaring a Pointer

dataType$ pointerName; ## Declaration of a pointer

### Accessing Memory Address

pointerName = &variableName;## Assigning the address of a variable to a pointer

### Dereferencing a Pointer

value = $pointerName; ## Accessing the value at the pointer's address



## main.joi - Example Program for Memory Addresses

#include <iostream>

using namespace std;

int joi() {

int number = 42; ## Original variable

int$ pointerNumber = &number; ## Pointer to the original variable

## Output the original value and its memory address

cout << "Original number: " << number << endl; ## Output: 42

cout << "Memory address of number: " << &number << endl; ## Output: (memory address)

cout << "Pointer points to value: " << $pointerNumber << endl; ## Output: 42

cout << "Memory address stored in pointer: " << pointerNumber << endl; ## Output: (memory address)

## Modify the original variable using the pointer

$pointerNumber = 100;

## Output the modified original value

cout << "Modified original number: " << number << endl; ## Output: 100

return 0; ## Indicate successful execution

}



# Function

## Function Declaration

returnType functionName(parameterType parameterName):

// Function body

:



## main.joi - Example Program for Functions

#include <iostream>

using namespace std;

## Function Declaration

int add(int a, int b):

return a + b; ## Function body to add two numbers

:

int joi() {

## Function Call

int sum = add(5, 7); ## Calling the function with arguments

## Output the result

cout << "The sum is: " << sum << endl; ## Output: 12

return 0; ## Indicate successful execution

}



## Multiple Parameters

returnType functionName(parameterType parameter1, parameterType parameter2, ...):

// Function body

:



## main.joi - Example Program for Functions with Multiple Parameters

#include <iostream>

using namespace std;

## Function Declaration

int calculateArea(int length, int width):

return length \* width; ## Function body to calculate area of a rectangle

:

int joi() {

## Function Call with multiple arguments

int area = calculateArea(5, 10); ## Calls calculateArea(5, 10)

## Output the result

cout << "The area of the rectangle is: " << area << endl; ## Output: 50

return 0; ## Indicate successful execution

}



## Return Keyword

Syntax

returnType functionName(parameters):

return value; ## Return a value from the function

:



## main.joi - Example Program for the Return Keyword

#include <iostream>

using namespace std;

## Function Declaration

int square(int number):

return number \* number; ## Function body to calculate the square of a number

:

int joi() {

## Function Call

int result = square(4); ## Calls square(4)

## Output the result

cout << "The square of 4 is: " << result << endl; ## Output: 16

return 0; ## Indicate successful execution

}



## Pass By Reference

returnType functionName(dataType& parameterName):

// Function body

:



## main.joi - Example Program for Pass By Reference

#include <iostream>

using namespace std;

## Function Declaration

void increment(int& value):

value++; ## Increment the original value

:

int joi() {

int number = 10; ## Original variable

## Output the original value

cout << "Original number: " << number << endl; ## Output: 10

## Function Call

increment(number); ## Pass by reference

## Output the modified value

cout << "Modified number after increment: " << number << endl; ## Output: 11

return 0; ## Indicate successful execution

}



## Variable Scope

### Local Scope

Syntax

void functionName() {

dataType variableName = value; ## Local variable

// Code that uses variableName

}



## main.joi - Example Program for Local Scope

#include <iostream>

using namespace std;

int joi() {

int localVar = 10; ## Local variable

cout << "Local variable: " << localVar << endl; ## Output: 10

return 0; ## Indicate successful execution

}



### Global Variable

dataType globalVariableName = value; ## Global variable

void functionName() {

// Code that uses globalVariableName

}



## main.joi - Example Program for Global Scope

#include <iostream>

using namespace std;

int globalVar = 20; ## Global variable

int joi() {

cout << "Global variable: " << globalVar << endl; ## Output: 20

return 0; ## Indicate successful execution

}

### Parameter Scope

void functionName(dataType parameterName):

// Code that uses parameterName

:



## main.joi - Example Program for Parameter Scope

#include <iostream>

using namespace std;

void display(int param):

cout << "Parameter value: " << param << endl; ## Output the parameter value

:

int joi() {

display(30); ## Function call with argument

return 0; ## Indicate successful execution

}

## Recursion

returnType functionName(parameters):

if (baseCondition):

return baseValue; ## Base case to terminate recursion

else:

return functionName(modifiedParameters); ## Recursive call

:



## main.joi - Example Program for Recursion

#include <iostream>

using namespace std;

## Recursive Function Declaration

int factorial(int n):

if (n <= 1):

return 1; ## Base case

else:

return n \* factorial(n - 1); ## Recursive call

:

int joi() {

int number = 5; ## Example input

int result = factorial(number); ## Function call

## Output the result

cout << "Factorial of " << number << " is: " << result << endl; ## Output: 120

return 0; ## Indicate successful execution

}



# OOPS

## 

## Class & Objects

class ClassName {

public:

// members

};



#include <iostream>

using namespace std;

class Car {

public:

void drive() {

cout << "Driving" << endl;

}

};

int joi() {

Car myCar; // Create an object of Car

myCar.drive(); // Call the method to drive

return 0; // Indicate successful execution

}



## Constructor

### Default Constructor

ClassName() { // Initialization }

}



### Parameterized Constructor

ClassName(Type parameter) {

// Initialization

}



#include <iostream>

using namespace std;

class Car {

public:

Car() {

cout << "Default Car created!" << endl;

}

Car(string model) { // Parameterized Constructor

cout << "Car created: " << model << endl;

}

};

int joi() {

Car car1; // Default Constructor

Car car2("Toyota"); // Parameterized Constructor

return 0; // Indicate successful execution

}



### Access Specifier

class ClassName {

private:

// private members

public:

// public members

};



#include <iostream>

using namespace std;

class Car {

private:

int speed;

public:

void setSpeed(int s) {

speed = s;

}

int getSpeed() {

return speed;

}

};

int joi() {

Car myCar; // Create an object of Car

myCar.setSpeed(100); // Assign value to the attribute

cout << "Speed: " << myCar.getSpeed() << " km/h" << endl; // Call the method to display speed

return 0; // Indicate successful execution

}



## Encapsulation

class ClassName {

private:

// private members

public:

void setData(Type value) {

// setter method

}

Type getData() {

// getter method

}

};



#include <iostream>

using namespace std;

class Car {

private:

int speed;

public:

void setSpeed(int s) {

speed = s;

}

int getSpeed() {

return speed;

}

};

int joi() {

Car myCar; // Create an object of Car

myCar.setSpeed(80); // Assign value to the attribute

cout << "Car speed: " << myCar.getSpeed() << " km/h" << endl; // Call the method to display speed

return 0; // Indicate successful execution

}



## Inheritance

class BaseClass {

// Base class members

};

class DerivedClass : public BaseClass {

// Derived class members

};



#include <iostream>

using namespace std;

class Vehicle {

public:

void start() {

cout << "Vehicle started" << endl;

}

};

class Car : public Vehicle {

public:

void drive() {

cout << "Car is driving" << endl;

}

};

int joi() {

Car myCar; // Create an object of Car

myCar.start(); // Call method from base class

myCar.drive(); // Call method from derived class

return 0; // Indicate successful execution

}



### 

### Multilevel Inheritance

class Base {

// Base class members

};

class Derived1 : public Base {

// Derived1 members

};

class Derived2 : public Derived1 {

// Derived2 members

};



#include <iostream>

using namespace std;

class Vehicle {

public:

void start() {

cout << "Vehicle started" << endl;

}

};

class Car : public Vehicle {

public:

void drive() {

cout << "Car driving" << endl;

}

};

class SportsCar : public Car {

public:

void accelerate() {

cout << "Sports car accelerating" << endl;

}

};

int joi() {

SportsCar mySportsCar; // Create an object of SportsCar

mySportsCar.start(); // Call method from Vehicle

mySportsCar.drive(); // Call method from Car

mySportsCar.accelerate(); // Call method from SportsCar

return 0; // Indicate successful execution

}



### Multiple Inheritance

class ClassA {

// Members of ClassA

};

class ClassB {

// Members of ClassB

};

class Derived : public ClassA, public ClassB {

// Members of Derived

};



#include <iostream>

using namespace std;

class Engine {

public:

void start() {

cout << "Engine started" << endl;

}

};

class Wheels {

public:

void rotate() {

cout << "Wheels rotating" << endl;

}

};

class Car : public Engine, public Wheels {

public:

void drive() {

start(); // Call method from Engine

rotate(); // Call method from Wheels

cout << "Car is driving" << endl;

}

};

int joi() {

Car myCar; // Create an object of Car

myCar.drive(); // Call the method to drive

return 0; // Indicate successful execution

}



### Inheritance Access

class Base {

protected:

int protectedVar;

public:

void setVar(int val) {

protectedVar = val;

}

};

class Derived : public Base {

public:

void showVar() {

cout << protectedVar << endl; // Accessible

}

};



#include <iostream>

using namespace std;

class Base {

protected:

int protectedVar;

public:

void setVar(int val) {

protectedVar = val;

}

};

class Derived : public Base {

public:

void showVar() {

cout << "Protected Variable: " << protectedVar << endl;

}

};

int joi() {

Derived obj; // Create an object of Derived

obj.setVar(10); // Set the protected variable

obj.showVar(); // Display the protected variable

return 0; // Indicate successful execution

}



## Polymorphism

### Compile-Time Polymorphism (Method Overloading)

class ClassName {

public:

void method(int);

void method(double);

};



#include <iostream>

using namespace std;

class Print {

public:

void show(int i) {

cout << "Integer: " << i << endl;

}

void show(double d) {

cout << "Double: " << d << endl;

}

};

int joi() {

Print printer; // Create an object of Print

printer.show(10); // Call method for integer

printer.show(10.5); // Call method for double

return 0; // Indicate successful execution

}

### Runtime Polymorphism (Method Overriding)

class Base {

public:

virtual void display();

};

class Derived : public Base {

public:

void display() override;

};



#include <iostream>

using namespace std;

class Base {

public:

virtual void display() { // Virtual function for runtime polymorphism

cout << "Display Base" << endl;

}

};

class Derived : public Base {

public:

void display() override { // Override the base class method

cout << "Display Derived" << endl;

}

};

int joi() {

Base$ b = new Derived(); // Create an object of Derived using Base pointer

b->display(); // Calls Derived's display

delete b; // Free memory

return 0; // Indicate successful execution

}

## Interfaces

Interface Interface {

public:

virtual void method() = 0; // Pure virtual function

};



#include <iostream>

using namespace std;

Interface IShape {

public:

virtual void draw() = 0; // Pure virtual function

};

class Circle : public IShape {

public:

void draw() override { // Override the pure virtual function

cout << "Drawing Circle" << endl;

}

};

int joi() {

Circle circle; // Create an object of Circle

circle.draw(); // Call the method to draw the circle

return 0; // Indicate successful execution

}



## //code for all OOPS concept

#include <iostream>

using namespace std;

class Print {

public:

void show(int i) {

cout << "Integer: " << i << endl;

}

void show(double d) {

cout << "Double: " << d << endl;

}

};

class Base {

public:

virtual void display() {

cout << "Display Base" << endl;

}

};

class Derived : public Base {

public:

void display() override {

cout << "Display Derived" << endl;

}

};

Interface IShape {

public:

virtual void draw() = 0; // Pure virtual function

};

class Circle : public IShape {

public:

void draw() override {

cout << "Drawing Circle" << endl;

}

};

int joi() {

// Compile-Time Polymorphism

Print printer; // Create an object of Print

printer.show(10); // Call method for integer

printer.show(10.5); // Call method for double

// Runtime Polymorphism

Base$ b = new Derived(); // Create an object of Derived using Base pointer

b->display(); // Calls Derived's display

delete b; // Free memory

// Interfaces

Circle circle; // Create an object of Circle

circle.draw(); // Call the method to draw the circle

return 0; // Indicate successful execution

}

int main() {

return joi();

}



# Exceptions

try:

// Code that may throw an exception

:

catch (ExceptionType e):

// Code to handle the exception

:



## main.joi - Example Program for Exception Handling

#include <iostream>

#include <string>

using namespace std;

public MyException: ## Custom exception class

str message; ## Message associated with the exception

MyException(str msg):

message = msg; ## Constructor to set the message

:

str what():

return message; ## Return the exception message

:

:

int divide(int a, int b):

if (b == 0):

throw MyException("Division by zero is not allowed."); ## Throw exception

:

return a / b; ## Perform division

:

int joi() {

try:

int result = divide(10, 0); ## Attempt to divide by zero

cout << "Result: " << result << endl;

:

catch (MyException e):

cout << "Exception caught: " << e.what() << endl; ## Handle the exception

:

return 0; ## Indicate successful execution

}



# Dynamic Mem Allocation

Syntax for Dynamic Memory Allocation

## Using new

dataType$ pointerName:

new dataType; ## Allocating memory

:



## Using delete

delete pointerName; ## Deallocating memory



## main.joi - Example Program for Dynamic Memory Allocation

#include <iostream>

#include <string>

using namespace std;

int joi() {

int$ intPtr: ## Pointer for dynamic allocation

intPtr = new int; ## Allocate memory for an integer

$intPtr = 42; ## Assign value

cout << "Value: " << $intPtr << endl; ## Print value

delete intPtr; ## Deallocate memory

// Dynamic array allocation

int size = 5; ## Size of the array

int$ arr: ## Pointer for dynamic array

arr = new int[size]; ## Allocate memory for an array

for (int i = 0; i < size; i++):

arr[i] = i + 1; ## Assign values

:

cout << "Array values: ";

for (int i = 0; i < size; i++):

cout << arr[i] << " "; ## Print array values

:

cout << endl;

delete[] arr; ## Deallocate array memory

return 0; ## Indicate successful execution

}



# Files

filePointer = fopen("filename", "mode"): ## Open file

:



fclose(filePointer): ## Close the opened file

:



fwrite(filePointer, dataPointer, size, count): ## Write data to the file

:



fread(dataPointer, size, count, filePointer): ## Read data from the file

:



## main.joi - Example Program for File Handling

int joi() {

// Open file for writing

file$ filePtr:

filePtr = fopen("example.txt", "w"): ## Open file in write mode

:

str data = "Hello, JOI File Handling!"; ## Data to write

// Write to the file

fwrite(filePtr, data.c\_str(), sizeof(char), data.length()): ## Write data

:

// Close the file

fclose(filePtr): ## Close the file

:

// Open file for reading

filePtr = fopen("example.txt", "r"): ## Open file in read mode

:

// Buffer to read data

char buffer[100]:

fread(buffer, sizeof(char), 100, filePtr): ## Read data into buffer

:

// Print the read data

cout << "Read from file: " << buffer << endl: ## Output the data

:

// Close the file

fclose(filePtr): ## Close the file

:

return 0: ## Indicate successful execution

}



# Type Casting

## Implicit Type Casting

dataType newVariable = existingVariable; ## Implicit conversion



## main.joi - Example Program for Implicit Type Casting

int joi() {

int intValue = 10: ## Integer value

float floatValue: ## Implicit conversion to float

floatValue = intValue: ## Implicitly converts int to float

cout << "Integer: " << intValue << ", Float: " << floatValue << endl: ## Output values

return 0: ## Indicate successful execution

}



## Explicit Type Casting

dataType newVariable = (dataType) existingVariable; ## Explicit conversion



## main.joi - Example Program for Explicit Type Casting

int joi() {

float floatValue = 10.5: ## Float value

int intValue: ## Explicit conversion to int

intValue = (int) floatValue: ## Explicitly converts float to int

cout << "Float: " << floatValue << ", Integer: " << intValue << endl: ## Output values

return 0: ## Indicate successful execution

}

