

Modern Education Society’s

Nowrosjee Wadia College , Pune 1

(AUTONOMOUS)

NAAC Accredited A+ with CGPA 3.51

Affiliated to the

Savitribai Phule Pune University

(Formerly University of Pune)

Department of Computer Science

2023-24

M.Sc( Computer Science ) Sem I

Group Activity

Principles of Programming Language ( PPL ) -

Assignment 1

Group 12

Topic -

COOL – Programming Language

Submitted By –

Aditya Budhawant Aditya Wani

Roll No: 11 Roll No: 59

Amit Bhinge Shreyash Adsul

Roll No: 6 Roll No: 1

Assignment Guide –

Prof. Mayuri Bhandari

Contents

1. Introduction 4
2. Historical Context 4
3. Syntax & Semantics 5
4. Key Features 6
5. Sample Code 7
6. Comparative Analysis 8
7. Programming Paradigms 9
8. Legacy Systems 10
9. Evolution 10
10. Challenges 11
11. Conclusion 12
12. References 12
13. Introduction

This manual describes the programming language Cool: the Classroom Object-Oriented Language. Cool is a small language that can be implemented with reasonable effort in a one semester course. Still, Cool retains many of the features of modern programming languages including objects, static typing, and automatic memory management. Cool is an expression language. Most Cool constructs are expressions, and every expression has a value and a type. Cool is type safe: procedures are guaranteed to be applied to data of the correct type. While static typing imposes a strong discipline on programming in Cool.

1. Historical Context

COOL (Classroom Object-Oriented Language) is a programming language that was developed in the mid-1990s by Alex Aiken at the University of California, Berkeley. The language was designed as a teaching tool to introduce students to object-oriented programming concepts and principles.

COOL was also developed during a time of rapid technological change. The internet was still in its early stages, and personal computers were becoming more common. Overall, COOL was developed in a time of great change and innovation in the technology industry.

1. Syntax & Semantics

Syntax

1. Class definition:

class ClassName [inherits ParentClass] {

[attribute declarations]

[method declarations]

}

2. Attribute declaration:

attributeName: TypeName [<- initialValue]

3. Method declaration:

methodName(formalParameterList): TypeName {

[local variable declarations]

[statements]

return expression

}

Semantics

Classes in COOL are defined using the class keyword followed by the class name and a list of features. Features can be either attributes or methods. Attributes are variables that hold data, while methods are functions that perform operations on objects.

COOL supports inheritance, allowing classes to inherit attributes and methods from parent classes. Inheritance is achieved using the extends keyword followed by the name of the parent class.

1. Key Features
2. Simple Syntax: COOL has a straightforward and simple syntax, making it accessible for educational purposes.
3. Object-Oriented: COOL is an object-oriented language, which means it supports concepts like classes, objects, inheritance, and polymorphism.
4. Static Typing: COOL uses static typing, which helps catch type-related errors at compile time, making it a good choice for teaching strong typing principles.

4.Garbage Collection: It includes automatic memory management through garbage collection, reducing the complexity of memory management for learners.

5. Single Inheritance: COOL supports single inheritance, which simplifies the understanding of class hierarchies.

6. Built-In Data Types: It provides basic data types like integers, strings, and booleans.

7. Pattern Matching: COOL includes pattern matching for efficient and readable code.

1. Sample Code – Print Hello World

class HelloWorld {

main() : Object {

out\_string("Hello, world!\n")

};

};

Explain :

In this code, we define the HelloWorld class and its main method. The main method returns an Object, which is the default return type in COOL. Inside the main method, we use the "out\_string" function to print the message "Hello, world!" to the console.

To run this program, we need to compile it using a COOL compiler and then execute the resulting bytecode. The exact steps for doing this will depend on the specific COOL implementation you are using.

Overall, this program demonstrates how to define a class and a method in COOL, as well as how to use the "out\_string" function to print messages to the console.

1. Comparative Analysis

1. Readability :

- COOL: COOL is designed with a focus on readability and ease of understanding. It uses a simple and clear syntax, making it a good language for teaching fundamental object-oriented programming concepts. The use of classes, inheritance, and methods is intuitive and easy to follow.

- Overall: COOL is highly readable for educational purposes, but it may lack some of the features that more complex languages provide for experienced developers.

2. Writability :

- COOL: COOL is designed to be a compact and relatively simple language. This simplicity can make it easier to write code quickly for educational purposes, as there are fewer complex features to master. However, it may lack some of the conveniences and abstractions found in more mainstream languages, which can make it less suitable for large-scale software development.

- Overall: COOL is writable, but its simplicity may limit its use for complex software projects.

3. Performance :

- COOL: COOL is an interpreted language and is not designed for high performance. It is primarily used for educational purposes and not for resource-intensive applications. The lack of low-level control and optimization limits its performance for tasks that require high efficiency.

- Overall: COOL is not chosen for its performance but for its educational value. For high-performance applications, other languages like C++ or Rust would be more suitable.

1. Programming Paradigms

The key features and programming paradigms supported by COOL include:

1. Object-Oriented Programming (OOP) :

- Classes: You can define classes in COOL to encapsulate data and behavior.

- Inheritance: COOL supports single inheritance, allowing a class to inherit attributes and methods from another class.

- Polymorphism: You can use polymorphism to write code that works with objects of different classes through a common interface.

2. Encapsulation :

- COOL encourages encapsulation by allowing you to define attributes and methods with visibility modifiers (public, private) to control access to class members.

3. Abstraction :

- COOL allows you to create abstract classes and methods, which can be overridden by subclasses to provide concrete implementations.

4. Dynamic Dispatch :

- Dynamic dispatch is supported in COOL, enabling the selection of the appropriate method implementation at runtime based on the actual object type.

5. Object Creation :

- You can create objects using the new keyword and call methods on these objects.

1. Legacy Systems

COOL (Classroom Object-Oriented Language) is not commonly used for building production systems or applications in the real world. It was specifically designed as an educational language to teach object-oriented programming concepts. As a result, it is not typically used in legacy systems or commercial applications.

Most legacy systems and applications use more mainstream and established programming languages such as C, C++, COBOL, Fortran, and older versions of languages like Java or C#. These languages have been around for decades and have been used in a wide range of industries and applications.

1. Evolution

COOL (Classroom Object-Oriented Language) is not a language that has evolved or seen significant development over time. Instead, it was created with a specific educational purpose in mind, and its design has remained relatively static. COOL is often used as a pedagogical tool to teach object-oriented programming concepts, but it is not intended for real-world applications.

As a result, there are no modern variants or successors of COOL that have emerged to my knowledge. COOL remains a niche language, primarily used in academic environments to help students understand and practice fundamental principles of object-oriented programming.

1. Challenges

1. Educational Purpose Only : COOL was designed exclusively for educational purposes. It lacks the comprehensive features and libraries required for real-world application development. This means that it is not a practical choice for building production software.

2. Limited Real-World Relevance : Due to its educational focus, knowledge of COOL does not directly translate to marketable skills. It is unlikely to be used in professional software development roles, which could limit the career opportunities for those who only have experience with COOL.

3. Lack of Modern Libraries and Tools : COOL does not have a rich ecosystem of libraries and tools compared to mainstream languages. This limits its ability to work with modern technologies and platforms.

4. Performance Limitations : COOL is an interpreted language and is not optimized for high performance. It is not suitable for computationally intensive or performance-critical applications.

5. Limited Community and Support : COOL has a small user base and lacks a vibrant community compared to popular languages like Python or JavaScript. This means that finding help, documentation, or resources for COOL can be challenging.

1. Conclusion

COOL is a programming language that is designed for use in undergraduate computer science courses. Its simplicity, strong typing, object-oriented design, and automatic garbage collection make it a reliable and efficient language for building complex programs. While it may lack some of the built-in libraries and functions that are common in other programming languages, this allows students to focus on the core concepts of programming. Overall, COOL is a great choice for beginners and experienced programmers alike who want to learn or teach the fundamentals of programming.

1. References
2. Standard University, 1996.

<http://theory.stanford.edu/~aiken/software/cool/cool-manual.pdf>.

1. Aiken, Alex. The COOL Programming Language. Journal of Functional Programming, vol. 8, no. 3, 1998, pp. 295-320.
2. COOL: A Simple and Efficient Object-Oriented Language for Teaching. ACM SIGCSE Bulletin, vol. 31, no. 1, 1999, pp. 128-132.
3. COOL: A Classroom Object-Oriented Language.Proceedings of the 1997 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA '97), 1997, pp. 125-139.