MSCS-632: Assignment 2: Lab Report: Syntax, Semantics and Memory Management

---

Sandesh Pokharel

University of the Cumberlands

MSCS-632 Advanced Programming Languages

Dr. Vanessa Cooper

February 2, 2025

Part 1: Analyzing Syntax and Semantics

Modified code in python

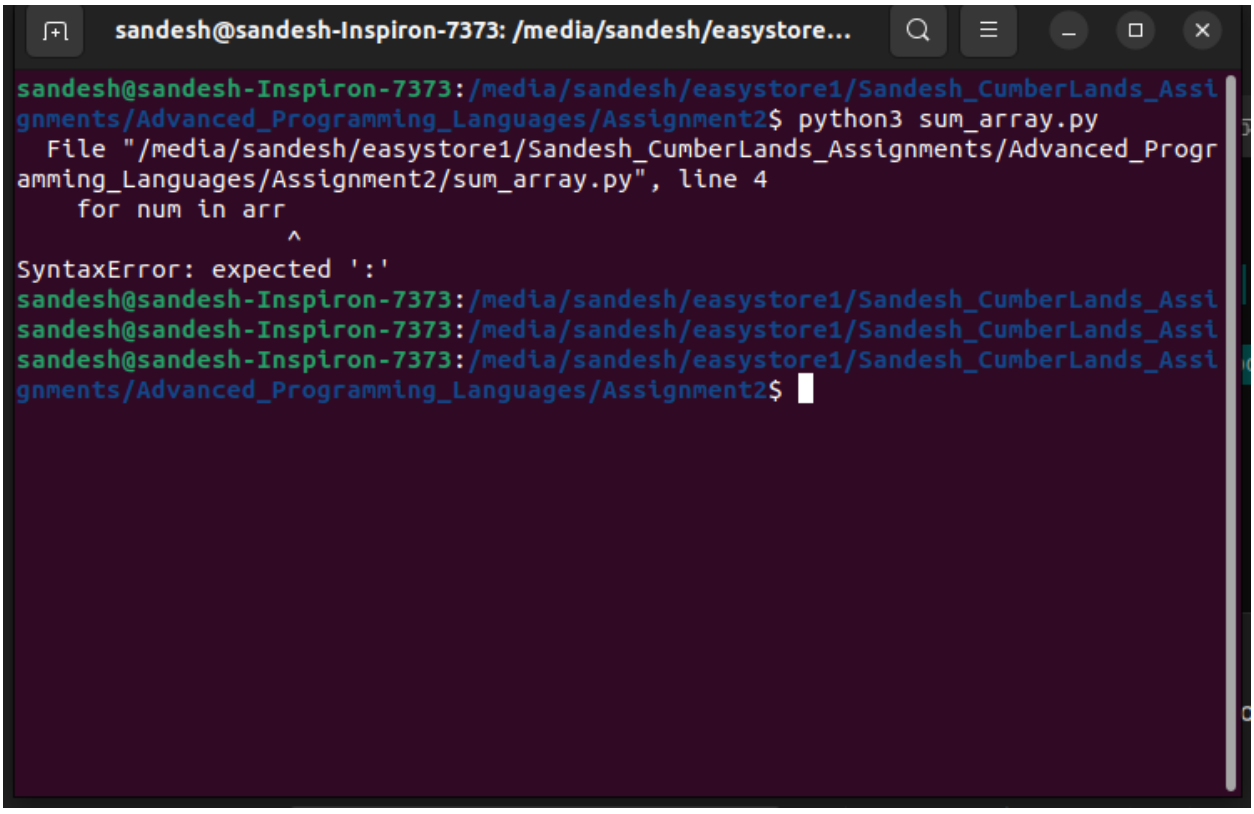
Activities Text Editor Feb

Open sum_
easystore /media/sandesh/easystore1/Sandesh..

```python
# Python: Calculate the sum of an array with syntax errors
def calculate_sum(arr) :   # Correct syntax
    total = o  # Syntax Error: 'o' should be '0' (invalid identifier)
    for num in arr
        total += num  # Syntax Error: Missing colon (:) after 'for' loop
    return total

numbers = [1, 2, 3, 4, 5]
result = calculate_sum (numbers)
print("Sum in Python :", result)
```

Result after running:



Explanation of the Error

Python expects a colon (:) at the end of control structures like loops, conditional statements, and function definitions. When the colon is missing, Python raises a SyntaxError, indicating where it expected the colon.

How Python Handles Syntax Errors

- **Immediate Error Detection:** Python stops execution immediately upon encountering a syntax error. It doesn't proceed to subsequent lines until the syntax issue is resolved.
- **Error Message Clarity:** The error message points directly to the line and position (^) where the problem occurred, making it easier to debug.

Comparison with Other Languages

- In **JavaScript**, a missing token like this might also cause a **SyntaxError**, but it may not be as strict about catching certain errors early (e.g., undefined variables might be flagged only when executed).
- In **C++**, the compiler performs a full syntax check before execution. Errors like missing parentheses or semicolons would prevent the program from compiling altogether.

Modified code in JavaScript

Execute | </> Source Code | Share | Help

```javascript
// JavaScript: Calculate the sum of an array with syntax errors
function calculateSum(arr) {
    let total = o;  // Syntax Error: 'o' is not defined
    for (let num of arr {  // Syntax Error: Missing closing parenthesis ')'
        total += num;
    }
    return total;
}

let numbers = [1, 2, 3, 4, 5];
let result = calculate Sum (numbers);  // Syntax Error: Unexpected space in function call
console.log("Sum in JavaScript:", result);
```

Results after running

```
/home/cg/root/67a047bb70b0f/script.js:4
    for (let num of arr {  // Syntax Error: Missing closing parenthesis ')'
                        ^

SyntaxError: Unexpected token '{'
    at internalCompileFunction (node:internal/vm:73:18)
    at wrapSafe (node:internal/modules/cjs/loader:1274:20)
    at Module._compile (node:internal/modules/cjs/loader:1320:27)
    at Module._extensions..js (node:internal/modules/cjs/loader:1414:10)
    at Module.load (node:internal/modules/cjs/loader:1197:32)
    at Module._load (node:internal/modules/cjs/loader:1013:12)
    at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:128:12
        )
    at node:internal/main/run_main_module:28:49

Node.js v18.19.1
```

Error message:
SyntaxError: Unexpected token '{'

Cause:

The error occurred because the **closing parenthesis ())** is missing in the for loop's declaration:
for (let num of arr { // Missing closing parenthesis ')'

How JavaScript Handles Syntax Errors:

- **Dynamic Detection:** JavaScript is interpreted dynamically, so syntax errors are caught when the script is executed.
- **Error Location:** JavaScript provides a detailed stack trace indicating the file name, line number (script.js:4), and where the error occurred (^ points to the unexpected { token).
- **Execution Halt:** The interpreter stops execution upon encountering a critical syntax error, similar to Python.

Comparison with Python and C++

- In **Python**, similar syntax errors, like a missing colon, would halt execution immediately with a clear **SyntaxError** message pointing to the error's location.
- In **C++**, the compiler would catch syntax issues like missing parentheses during the compilation stage, preventing the program from running until all errors are resolved.

Modified C++ code

```cpp
#include <iostream>
using namespace std;

int calculateSum(int arr[], int size) {
    int total = o;  // Syntax Error: 'o' is not a valid integer literal
    for (int i = o; i < size; i++ {  // Syntax Error: Missing closing parenthesis ')'
        total += arr[i];
    }
    return total;
}

int main () {
    int numbers [] = {1, 2, 3, 4, 5};
    int size = sizeof(numbers) / sizeof( numbers [o]);  // Syntax Error: 'o' is not valid
    int result = calculateSum(numbers, size);
    cout << "Sum in C++" " << result << endl;  // Syntax Error: Missing operator between strings
    return o;  // Syntax Error: 'o' is not defined
}

```

Error after running

```
main.cpp:16:26: error: missing terminating " character
   16 |     cout << "Sum in C++" " << result << endl;  // Syntax Error: Missing
      operator between strings
      |
         ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
main.cpp: In function 'int calculateSum(int*, int)':
main.cpp:5:17: error: 'o' was not declared in this scope
    5 |     int total = o;  // Syntax Error: 'o' is not a valid integer literal
      |                 ^
main.cpp:6:34: error: expected ')' before '{' token
    6 |     for (int i = o; i < size; i++ {  // Syntax Error: Missing closing
       parenthesis ')'
      |         ~                       ^~
      |                                 )
main.cpp: In function 'int main()':
main.cpp:14:51: error: 'o' was not declared in this scope
   14 |     int size = sizeof(numbers) / sizeof( numbers [o]);  // Syntax Error: 'o'
      is not valid
      |                                                   ^
main.cpp:16:25: error: expected ';' before 'return'
   16 |     cout << "Sum in C++" " << result << endl;  // Syntax Error: Missing
      operator between strings
      |                         ^
      |                         ;
   17 |     return o;  // Syntax Error: 'o' is not defined
      |     ~~~~~~
```

Explanation of Errors and Warnings

1. Error 1: Missing terminating string
2. Error 2: Undeclared variable 'o
3. Error 3: Missing closing parenthesis in the for loop
4. **Error 4: 'o' not declared in the array indexing**
5. Error 5: Missing Semicolon
6. Error 6: Undeclared variable in return statement

How C++ Handles Syntax Errors

- **Compile-Time Detection:** C++ uses a static type system and requires all syntax errors to be resolved during compilation. The program won't compile until all errors are fixed.
- **Detailed Error Reporting:** C++ provides specific error messages indicating the line number, issue description, and sometimes a suggestion (e.g., expected ')').
- **Multiple Errors at Once:** Unlike Python and JavaScript, C++ can report multiple syntax errors in one compilation pass.

Part 2: Memory Management

Introduction

This report analyzes dynamic memory management in Rust, Java, and C++ through three small programs. Each program demonstrates the language's approach to allocation, deallocation, and error handling. Key concepts such as ownership, garbage collection, and manual memory management are explored.

Github link for files: https://github.com/sanspokharel26677/MSCS-632-Assignment2

1. Rust: Ownership and Borrowing

Code Explanation (memory_management_rust.rs)

In the Rust program, a dynamically allocated vector is created using the vec![] macro. The vector is modified by borrowing it mutably through a function call.

Memory Management Approach

- **Ownership Model:** Rust enforces strict rules that each value has a single owner at a time.

- **Borrowing:** Data can be temporarily borrowed by functions, either mutably or immutably.
- **Automatic Deallocation:** When the owner goes out of scope, Rust automatically deallocates the memory, preventing memory leaks.
- **Error Prevention:** Rust's compiler ensures memory safety by preventing dangling pointers and invalid memory access at compile-time.

Advantages:

- Eliminates common memory errors like use-after-free and dangling pointers.
- Ensures deterministic memory deallocation.

Disadvantages:

- The ownership model may introduce complexity for developers new to Rust.

2. Java: Garbage Collection

Code Explanation (MemoryManagement.java)

In the Java program, an ArrayList is dynamically allocated and filled with integers. The program demonstrates garbage collection by clearing the list and optionally invoking System.gc().

Memory Management Approach

- **Automatic Allocation and Deallocation:** Objects are dynamically allocated on the heap.
- **Garbage Collection:** The Java Virtual Machine (JVM) automatically reclaims memory when objects are no longer reachable.
- **Memory Safety:** Java prevents issues like dangling pointers by abstracting low-level memory operations.

Advantages:

- Reduces developer responsibility for managing memory.

- Prevents memory leaks in most scenarios.

Disadvantages:

- Garbage collection can introduce performance overhead, particularly in latency-sensitive applications.
- Memory deallocation timing is non-deterministic.

3. C++: Manual Memory Management

Code Explanation (memory_management.cpp)

In the C++ program, an array is dynamically allocated using the new keyword. The program demonstrates manual deallocation using delete[].

Memory Management Approach

- **Manual Allocation:** Memory is allocated using new and must be explicitly deallocated using delete[].
- **Prone to Errors:** Improper handling can lead to memory leaks or dangling pointers if memory is not freed or is accessed after being freed.

Advantages:

- Provides full control over memory allocation and deallocation.
- Can be optimized for performance in resource-constrained environments.

Disadvantages:

- Increased risk of memory errors, such as leaks and segmentation faults.
- Requires careful management of pointers and allocation lifecycle.