CodInClub

Powered By BridgeLabz

Wk5 /S4/ Lecture # : DSOOPS-23

Revision:

Easy Problems (3)

1. Pointer Operations and Address Manipulation Write a C++ program that:

- Declares three integer variables (a=10, b=20, c=30)
- Creates pointers to each variable
- Swaps the values of the variables using only pointer operations (no direct variable access)
- Implements a function `findLargest` that takes three integer pointers as parameters and returns a pointer to the variable containing the largest value
- Prints the original addresses, values, and final values after swapping
- Uses the returned pointer from `findLargest` to modify the largest value to 100 and display all variables again

2. Simple Function Overloading Create three overloaded functions named `calculate` that:

- Takes two integers and returns their sum
- Takes two floats and returns their product
- Takes three integers and returns their average as a float Write a main function to test all three versions.

3. Call by Value vs Call by Reference Write two functions: one that takes an integer by value and tries to double it, and another that takes an integer by reference and doubles it. Demonstrate the difference by calling both functions from main and showing the results.

Medium Problems (2)

4. Pointer Arithmetic and Array Access Create a program that declares an array of 5 integers, uses a pointer to traverse the array and calculate the sum of all elements. Then modify the program to use pointer arithmetic to access array elements instead of array indexing. Also implement a function that takes the array by pointer and returns the sum.

5. Function Overloading with Different Parameter Types Design a class `Calculator` with overloaded `process` functions that can handle:

- Two integers (returns sum)
- Two pointers to integers (returns sum of values pointed to)
- An integer by reference (doubles the value and returns it)
- An integer by value and an integer by pointer (returns their product) Demonstrate all variations and show how the compiler resolves the correct function.

Hard Problem (1)

6. Smart Pointer Implementation and Memory Management Implement a simplified version of a smart pointer class called `SmartPtr` that:

- Uses reference counting to manage memory automatically
- Supports copy constructor and assignment operator with proper reference counting
- Provides `get()`, `reset()`, and `use_count()` methods
- Automatically deletes the managed object when reference count reaches zero
- Overloads `*` and `->` operators for accessing the managed object

Then write a test program that demonstrates:

- Creating multiple SmartPtr objects pointing to the same data
- Showing reference count changes as pointers are created/destroyed
- Comparing behavior with raw pointers in terms of memory management
- Function calls using SmartPtr with call by value, reference, and pointer semantics

Include proper error handling and demonstrate how your SmartPtr prevents memory leaks compared to raw pointers.