Introduction to Computer Programming

- Week 8&9

- *Eng. Sylvain Manirakiza -*

Sun - March 23, 2025

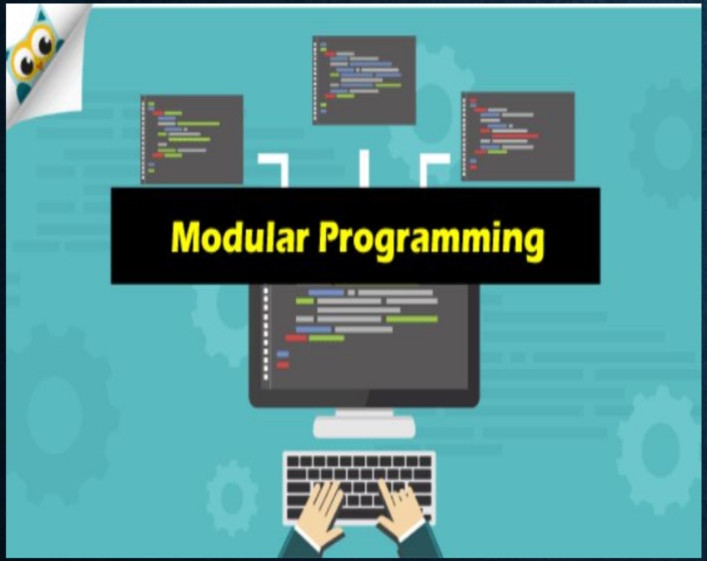
Chap 5. Pointers and Memory management

- Week 8&9

- *Eng. Sylvain Manirakiza* -

Sun - March 23, 2025

Quick recap of Chap 4



- Modular programming helps structure and organize code.
- **Predefined functions** save time and improve efficiency.
- **User-defined functions** allow reusability.
- **Recursion** is useful for solving complex problems using smaller steps.

+ 2 For a quick show of

- ❑ **Recursion** is useful for solving complex problems using smaller steps.

Exercices - Modular pseudocode

1. Write a Function to Calculate the Area of a Rectangle and displays the result.
2. Write a function **is_even(n)** that checks if a number is even. The main program should take input from the user and call the function.
3. Write a modular pseudocode program that converts Celsius to Fahrenheit using a function.
4. Write a recursive function that calculates the factorial of a number.
5. Write a function that takes three numbers and returns the largest.

Exercices #1 - Responses

```
Function calculate_area(length, width)
  area ← length * width
  Return area
End Function

Start
  Write "Enter length: "
  Read length
  Write "Enter width: "
  Read width

  result ← calculate_area(length, width)
  Write "The area of the rectangle is: ", result
End
```

Exercices #2 - Responses

```
Function is_even(n)
  If n MOD 2 = 0 Then
    Return "Even"
  Else
    Return "Odd"
  End If
End Function

Start
  Write "Enter a number: "
  Read num

  result ← is_even(num)
  Write "The number is: ", result
End
```

Exercices #3 - Responses

```
Function celsius_to_fahrenheit(celsius)
  fahrenheit ← (celsius * 9/5) + 32
  Return fahrenheit
End Function

Start
  Write "Enter temperature in Celsius: "
  Read tempC

  tempF ← celsius_to_fahrenheit(tempC)
  Write "Temperature in Fahrenheit: ", tempF
End
```

Exercices #4 - Responses

```
Function factorial(n)
  If n = 1 Then
    Return 1
  Else
    Return n * factorial(n - 1)
  End If
End Function

Start
  Write "Enter a number: "
  Read num

  result ← factorial(num)
  Write "Factorial of ", num, " is: ", result
End
```

Exercices #5 - Responses

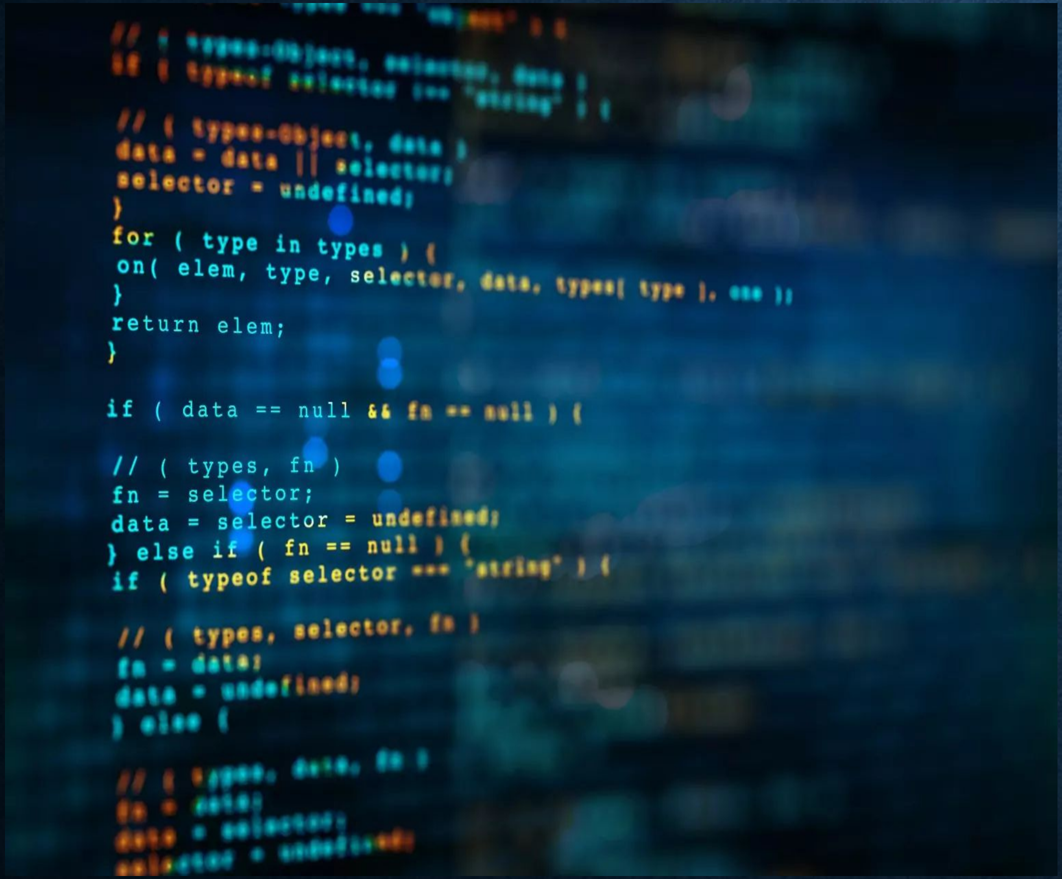
```
Function find_max(a, b, c)
  If a >= b AND a >= c Then
    Return a
  Else If b >= a AND b >= c Then
    Return b
  Else
    Return c
  End If
End Function
```

```
Start
  Write "Enter first number: "
  Read num1
  Write "Enter second number: "
  Read num2
  Write "Enter third number: "
  Read num3

  max_value ← find_max(num1, num2,
num3)
  Write "The largest number is: ",
max_value
End
```

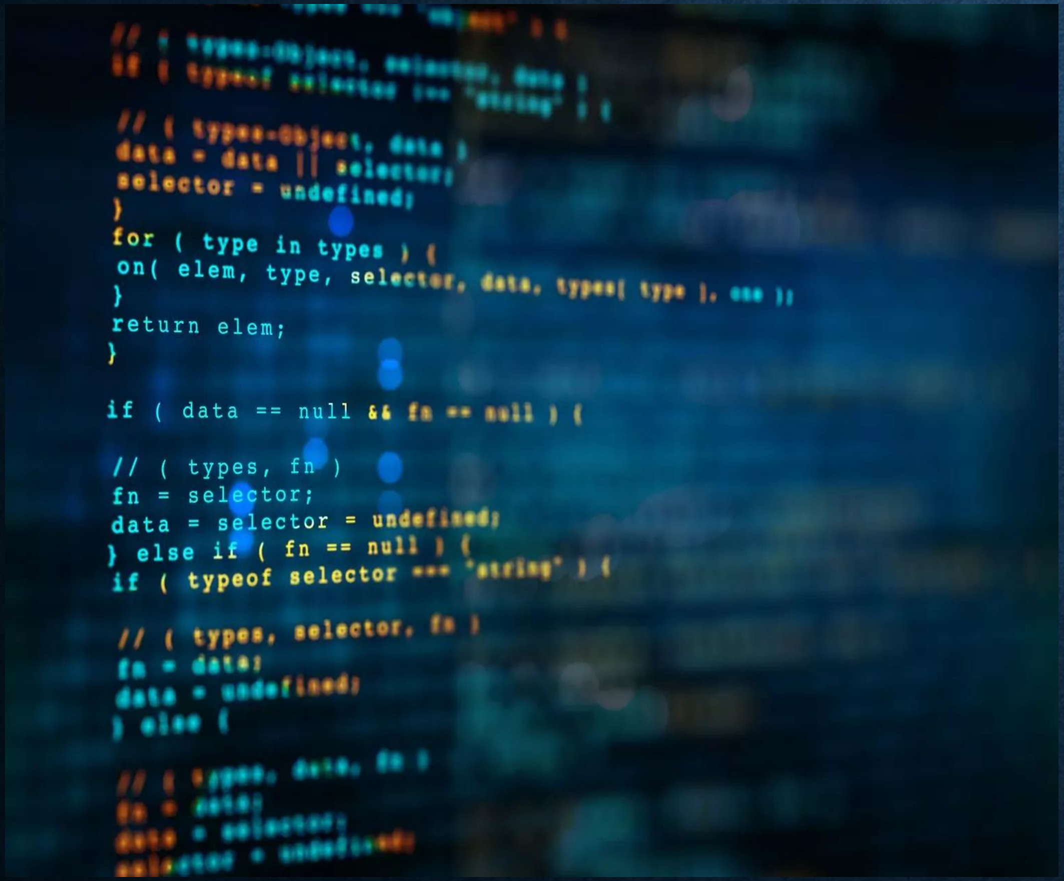
Chap 5: Pointers and Memory Management

What is a Pointer?



- A pointer is a variable that stores the memory address of another variable.Unlike normal variables it does not store user given or processed value, instead **it stores valid computer memory address**.
- Think of it as a "reference" to a location in memory.

Why Use Pointers?



- Direct memory access for efficiency.
- Dynamic memory allocation.
- Enables complex data structures like linked lists, trees, and graphs.

Memory Basics

- **Memory Addresses**
  - Every variable is stored in memory at a unique address.
  - *Example: int X = 5; → x is stored at address 1000.*

Memory Basics

- **Visualization**

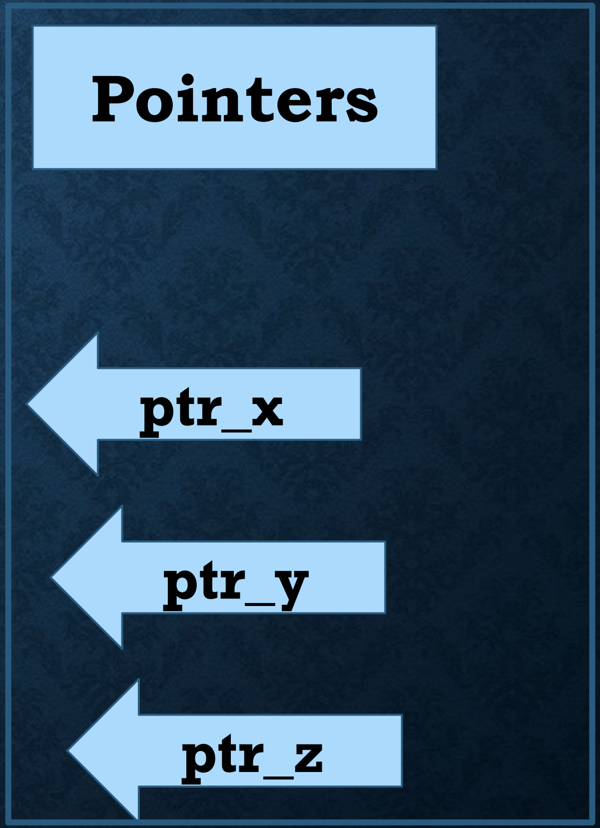
Memory		
Address	Variable	Value
1000	X	5
1004	Y	10
1008	Z	15

In this diagram:

- x is stored at address 1000 with a value of 5.
- y is stored at address 1004 with a value of 10.
- z is stored at address 1008 with a value of 15.

Pointers in Memory

Memory		
Address	Variable	Value
1000	X	5
1004	Y	10
1008	Z	15



Pointers in Memory

Memory		
Address	Variable	Value
1000	X	5
1004	Y	10
1008	Z	15
1012	ptr_x	1000
1016	ptr_y	1004
1020	ptr_z	1008

Pointers in Memory

- ptr_x is a pointer that points to the address **1000**, where the variable x is stored.
- ptr_y is a pointer that points to the address **1004**, where the variable y is stored.
- ptr_z is a pointer that points to the address **1008**, where the variable z is stored.

Declaring and Using Pointers

- **Assigning Addresses / Referencing**
  - Use the & operator to get the address of a variable.
  - *Example: SET ptr TO &X or p <—&X*
- **Dereferencing Pointers**
  - Use the * operator to access the value at the address.
  - *Example: PRINT "Value at address stored in ptr: ", or*
  - *Write *ptr*

Declaration and Using Pointers

- You can access a variable in two ways.
  - Direct Access: *You can directly use the variable name to access the variable.*
  - Indirect Access: *You use a pointer to access that variable.*

```
declare int a, *ptr;          //declaration
ptr=&a;                       //initialization
  write ("direct access("Direct Access, a=a);
  write("Indirect Access, a=*ptr);
```

Pseudocode Example

- *Variable y ← 10*
- *Pointer ptr_y ← **&y***   *// store address of y in ptr*
- *Write "Address stored in ptr_y: ", ptr_y*
- *Write "Value pointed by ptr_y: ", ***ptr_y***

Output:
Address stored in ptr_y: 1004
Value pointed by ptr_y: 10

Pointer Example in Pseudo-code

- *DECLARE X AS INTEGER*
- *DECLARE ptr AS POINTER TO INTEGER*

- *SET X TO 42*
- *SET ptr TO &X*

- *PRINT "Value of X: ", X*
- *PRINT "Address of X: ", &X*
- *PRINT "Value at address stored in ptr: ", *ptr*

Dynamic Memory Allocation

- **What is Dynamic Memory Allocation?**
  - Allocating memory at **runtime** instead of **compile time**.
  - Useful when the size of data is unknown beforehand.

Dynamic Memory Allocation

- Pseudo-code for Allocation
  - Use **ALLOCATE** and **DEALLOCATE** keywords.
  - *Example:*

Dynamic Memory Allocation -Example

```
DECLARE ptr AS POINTER TO INTEGER
ALLOCATE ptr
SET *ptr TO 100
PRINT "Value at allocated memory: ", *ptr
DEALLOCATE ptr
```

Memory Management

- **Why Manage Memory?**
  - Prevent memory leaks (unused memory not deallocated).
  - Avoid dangling pointers (pointers to deallocated memory).
- **Best Practices**
  - Always deallocate memory after use.
  - Set pointers to NULL after deallocation.

Common Pointer Errors

- **Null Pointers**
  - Pointers that do not point to any valid memory location.
- **Dangling Pointers**
  - Pointers that reference deallocated memory.
- **Memory Leaks**
  - Forgetting to deallocate memory.

Example of Memory Leak in Pseudo-code

DECLARE ptr AS POINTER TO INTEGER

ALLOCATE ptr

*SET *ptr TO 50*

// Forgot to DEALLOCATE ptr → Memory Leak!

Pointers and Arrays

Arrays as Pointers

- An array name is essentially a pointer to its first element.
- Example:

Example of Memory Leak in Pseudo-code

```
DECLARE arr[3] AS INTEGER
SET arr[0] TO 10
SET arr[1] TO 20
SET arr[2] TO 30
DECLARE ptr AS POINTER TO INTEGER
SET ptr TO arr
PRINT "First element: ", *ptr
PRINT "Second element: ", *(ptr + 1)
```

Pointers and Functions

Passing Pointers to Functions

- Allows functions to modify the original data
- *Example(Next Slide):*

Pointers and Functions

```
FUNCTION increment(ptr AS POINTER TO INTEGER)
    SET *ptr TO *ptr + 1
END FUNCTION


DECLARE X AS INTEGER
SET X TO 5
CALL increment(&X)
PRINT "Value of X after increment: ", X
```

Recap

1. Pointers store memory addresses.
2. Dynamic memory allocation allows runtime memory management.
3. Proper memory management prevents leaks and errors.
4. Pointers enable powerful programming techniques.

Applications

Write pseudocode to:

1. Dynamically allocate space for 4 numbers
2. Read values into memory using a pointer
3. Calculate and print their total sum
4. Free the memory

Pseudocode Solution

```
Start
  Pointer p ← Allocate(4)     // Allocate space for 4 numbers
  sum ← 0
  For i ← 0 to 3 Do
    Write "Enter number ", i+1, ":"
    Read p[i]                  // Store input at allocated address
    sum ← sum + p[i]
  End For
  Write "Total sum is: ", sum
  Free(p)                      // Release the allocated memory
End
```

Exercises

1. Write pseudo-code to swap two numbers using pointers.
2. Write a function that accepts a pointer to an array of 3 integers and returns the sum.

Responses - #1

```
Function swap(aPointer, bPointer)
  temp ← *aPointer
  *aPointer ← *bPointer
  *bPointer ← temp
End Function

Start
  Variable x ← 10
  Variable y ← 20

  Pointer px ← &x
  Pointer py ← &y

  Write "Before swap: x = ", x, ", y = ", y
  Call swap(px, py)
  Write "After swap: x = ", x, ", y = ", y
End
```

Responses - #2

```
Function sumArray(p)
  sum ← 0
  For i ← 0 to 2 Do
    sum ← sum + p[i]
  End For
  Return sum
End Function

Start
  Pointer arr ← Allocate(3)
  arr[0] ← 3
  arr[1] ← 5
  arr[2] ← 7

  result ← sumArray(arr)
  Write "Sum is: ", result

  Free(arr)
End
```

Find more Pointers and Memory management [Group exercises here]

End