Introduction to Computer Science – D684

Section 3

Lesson 1

* 1. **– Common Fundamental Algorithms**

|  |  |  |  |
| --- | --- | --- | --- |
| Category | Type | Definition | Example |
| Algorithms with **Selection** | **Selection** | Uses decision making (conditional statements) to choose between different actions based on a condition | If-Else Example  Determine if a number is even or odd:  if (num % 2 == 0)  print (“Even”);  else print (“Odd”); |
| Algorithms with **Repetition** | Repetition types below | Executes a block of code multiple times using loops | See below examples |
| **Count Controlled Loops** | A loop that repeats a set number of times | For Loop Example:  Print numbers 1 to 10:  for i in range(1,11): print(i) |
| **Event Controlled Loops** | A loop that runs until a specific event occurs (eg user input, condition met) | While Loop Example:  Keep asking for input until user enters ‘exit’:  while (input( ) != ”exit”): print(“Try Again”) |
| **Square Root Algorithms** | Used to find the square root of a number, typically through iterative approximation or mathematical functions | Newton’s Method Example:  x = number guess = x/2 while abs(guess \* guess -x) > 0.0001: guess = (guess + x / guess) / 2 print(guess) |

**Abstract step** = step that needs to be expanded further

**Concrete step** = step that does not need to be expanded

\*\*\* Arrays and Records covered in Section 1 Lesson Notes\*\*\*

**Composite Variables** = data structures that hold multiple values (eg arrays and records)

**Searching Algorithms**

|  |  |  |
| --- | --- | --- |
| Feature | **Sequential (Linear) Search** | **Binary Search** |
| Definition | Searches for an item by checking each element one by one from start to finish | Divides a sorted array in half repeatedly to find the target |
| Sorted or Unsorted Data? | **Unsorted** or **sorted** | **Sorted** only |
| Best at | Searching small or unsorted lists since no sorting is required | Fast searching on **large**, sorted datasets due to logarithmic efficiency |
| Worst at | Large datasets, as it requires checking every element in the worst case | Unsorted data, as it requires sorting beforehand, which can be expensive |

**Sorting Algorithms**

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Definition** | **Best At** | **Worst At** |
| **Selection sort** | Repeatedly finds the smallest element and swaps it with the current position | Minimizing swaps, since it makes at most n swaps, which can be useful in scenarios where swapping is expensive (e.g., writing to disk) | Dynamically changing data, since it does not adapt well to newly inserted elements and must be fully re-run |
| **Bubble sort** | Repeatedly swaps adjacent elements if they are in the wrong order, pushing the largest element to the end in each pass | Detecting if an array is already sorted, since it can finish early with an optimized version that stops after a full pass with no swaps | Swapping-intensive operations, as it makes many unnecessary swaps, making it slow in practice |
| **Insertion sort** | Builds the sorted list one element at a time, inserting each element into the correct position | Handling small real-time data, such as when new elements are continuously added and need to be sorted (e.g., keeping a leaderboard sorted) | Sorting data with many large elements near the beginning, as shifting many large elements multiple positions slows it down |
| **Quick sort** | Selects a pivot, partitions the array around the pivot, and recursively sorts subarrays | Handling large datasets efficiently, as it quickly partitions and sorts data, making it much faster than simpler sorting algorithms | Sorting nearly sorted or highly unbalanced data, since poor pivot choices can cause excessive recursive calls and slow performance |

Lesson 2

**2.1– Pseucode for Arithmetic**

Programmers Use 2 Primary Tools for Logic Planning:

1. Pseudocode = pseudo meaning false and code, sentences that are similar to written computer language but do no follow syntax rules, typical standards:
   1. Beginning statement = start, begin
   2. Ending statement = stop, end
   3. Statements are indented a few spaces
   4. Each program statement appears on single line if possible. When not possible, continuation lines are indented
   5. Statements begin with lowercase letters
   6. No punctuation is used to end a statement
   7. Each program statement performs one action (eg input, processing, or output)
   8. Module specific standards:
      1. Whenever a module name is used, it is followed by a set of parentheses
      2. Begin with the module name and end with ‘return’; these are always aligned
2. Flowchart = pictorial representation of the same logical steps
   1. Input/Output (I/O Symbol) = parallelogram
   2. Processing symbol = rectangle
   3. Decision = diamond
   4. Flowlines or arrows = arrows to connect the steps
   5. Terminal symbols (start/stop) = lozenge

A diagram of different types of objects

AI-generated content may be incorrect.Programming Logic and Design by Joyce Farrell Figure 1-7 pg 15

**Arithmetic Operations in Pseudocode**

|  |  |  |
| --- | --- | --- |
| Operation | Pseudocode Representation | Example |
| Addition | sum = a + b | a = 5, b = 3 → sum = 8 |
| Subtraction | difference = a - b | a = 10, b = 4 → difference = 6 |
| Multiplication | product = a \* b | a = 6, b = 7 → product = 42 |
| Division | quotient = a / b | a = 20, b = 4 → quotient = 5 |

**Combining Arithmetic Operations**

|  |  |  |
| --- | --- | --- |
| Scenario | Pseudocode Representation | Example Output |
| Calculate Average | sum = a + b + c  average = sum / 3 | a = 10, b = 20, c = 30 → average = 20 |
| Calculate Area of a Rectangle | area = length \* width | length = 10, width = 5 → area = 50 |
| Calculate Final Price After Discount & Tax | discount amount = original price \* discount rate  discounted price = original price - discount amount  sales tax = discounted price \* sales tax rate  final price = discounted price + sales tax | original price = 100, discount rate = 0.20, sales tax rate = 0.05  discount amount = 20  discounted price = 80  sales tax = 4  final price = 84 |

**Practice Problems & Possible Solutions**

|  |  |  |
| --- | --- | --- |
| Problem | Pseudocode Representation | Example Input/Output |
| Calculate Perimeter of a Rectangle | length = input  width = input  perimeter = 2 \* (length + width)  output perimeter | **Input:** length = 10, width = 5  **Output:** perimeter = 30 |
| Determine Total Cost of Items in Shopping Cart | price\_per\_item = input  quantity = input  total\_cost = price\_per\_item \* quantity  output total\_cost | **Input:** price\_per\_item = 20, quantity = 3  **Output:** total\_cost = 60 |
| Compute Simple Interest | principal = input  interest\_rate = input  time = input  simple\_interest = principal \* interest\_rate \* time  output simple\_interest | **Input:** principal = 1000, interest\_rate = 0.05, time = 3  **Output:** simple\_interest = 150 |

**2.2– Floating-Point Numbers, Functions, and Other Topics**

|  |  |  |
| --- | --- | --- |
| Concept | Description | Example Pseudocode |
| Floating-Point Numbers (Floats) | Numbers with decimal points, used for fractional values and scientific calculations. May introduce rounding errors due to limited precision | pi = 3.14159  radius = 2.5  area = pi \* (radius ^ 2) |
| Integer Division  // | Division of two integers where the quotient is an integer, discarding any remainder. Useful for distributing items evenly | total\_items = 17  items\_per\_box = 4  boxes\_needed = total\_items // items\_per\_box |
| Type Conversion | Changing a value from one data type to another.  Includes **implicit conversion** (automatic) and **explicit conversion** (manual) | **Implicit:**  num\_int = 10  num\_float = 2.5  result = num\_int + num\_float  **Explicit:**  num\_str = "123"  num\_int = int(num\_str)  num\_float = 9.99  num\_int = int(num\_float) |

Implicit vs. Explicit Type Conversion

|  |  |  |
| --- | --- | --- |
| Type | Description | Key Difference |
| **Implicit Conversion**  Smaller → Larger | Automatically performed by the compiler when no data loss occurs. Happens when combining different data types (e.g., int + float) | - No manual intervention required.  - No data loss (**smaller** to **larger** type).  - Happens automatically during operations. |
| **Explicit Conversion**  Larger → Smaller OR  Between types that do not auto-convert | Manually performed by the programmer using type conversion functions (e.g., int(), float(), str()) | - Requires a specific function call.  - Possible data loss (e.g., float to int).  - Used when conversion doesn’t happen automatically. |

**Mathematical Functions**

|  |  |  |  |
| --- | --- | --- | --- |
| Function | Description | Example in Pseudocode | Use Case |
| sqrt() | Calculates the square root of a given number | distance = sqrt(x^2 + y^2) | Used in geometry, physics, and graphics for calculating distances or roots of numbers |
| abs() | Returns the absolute value of a given number, removing any negative sign | difference = abs(a - b) | Ensures results are always nonnegative; useful in calculations involving magnitude, such as differences |
| round() | Rounds a floating-point number to a specified number of decimal places or to the nearest integer if no precision is given | rounded\_value = round(value, 2) | Common in financial calculations, data analysis, or formatting where rounded numbers are needed |

Lesson 3

**3.1– Introduction to Programming**

Structure = basic unit of programming logic

3 Basic Structures

1. Sequence
2. Selection
3. Loop

|  |  |  |  |
| --- | --- | --- | --- |
| Structure Type | Definition | Key Components | Example |
| Sequence Structure | The simplest control structure, where instructions are executed in order, one after another | Step-by-step execution of statements | a = 5 b = 10 sum = a + b print(sum) |
| Selection/Decision Structure | A control structure that allows a program to choose between different paths based on a condition | **End Structure Statement** - Marks the end of a control structure **Dual-Alternative Selection** - Executes one block if a condition is true, another if false  (if, then…else) **Single-Alternative Selection** - Executes a block only if the condition is true, otherwise skips it  (if,then) **Null Case** - A decision where nothing happens if a condition is false | **Dual-Alternative Selection:**  IF age >= 18 THEN PRINT "You can vote" ELSE PRINT "You cannot vote" ENDIF  **Single-Alternative Selection:**  IF temp > 100 THEN PRINT "Warning: Overheating!" ENDIF  **Null Case:**  IF score >= 50 THEN PRINT "You passed!" ENDIF |
| Loop Structure | A control structure that repeats a block of code while a condition remains true | **Loop Body** - The set of instructions executed in each iteration **While Loop** - A loop that runs as long as a condition remains true | **Loop Body & While Loop:** count = 1 WHILE count <= 15 PRINT count count = count + 1 ENDWHILE |

Other Key Terms

|  |  |  |  |
| --- | --- | --- | --- |
| Term | Definition | Key Concept | Example |
| Stacking Structure | The process of placing multiple control structures one after another, ensuring they execute in order | Uses multiple structures in sequence without nesting | INPUT name PRINT "Hello, " + name INPUT age IF age >= 18 THEN PRINT "You are an adult." ELSE PRINT "You are a minor." ENDIF |
| Nesting Structure | Placing one control structure inside another, allowing for complex decision-making and looping | A structure inside another structure (e.g., loops inside loops or IF inside loops) | WHILE score < 100 INPUT score IF score >= 50 THEN PRINT "Pass" ELSE PRINT "Fail" ENDIF ENDWHILE |
| Block | A group of statements treated as a single unit, typically enclosed within an IF statement, loop, or function | A set of instructions grouped under a control structure | FUNCTION calculate\_area(radius) area = 3.14 \* radius \* radius RETURN area END FUNCTION |