Problem Solving Technique

Lesson 1: Introduction to Program development with Algorithm and pseudocode

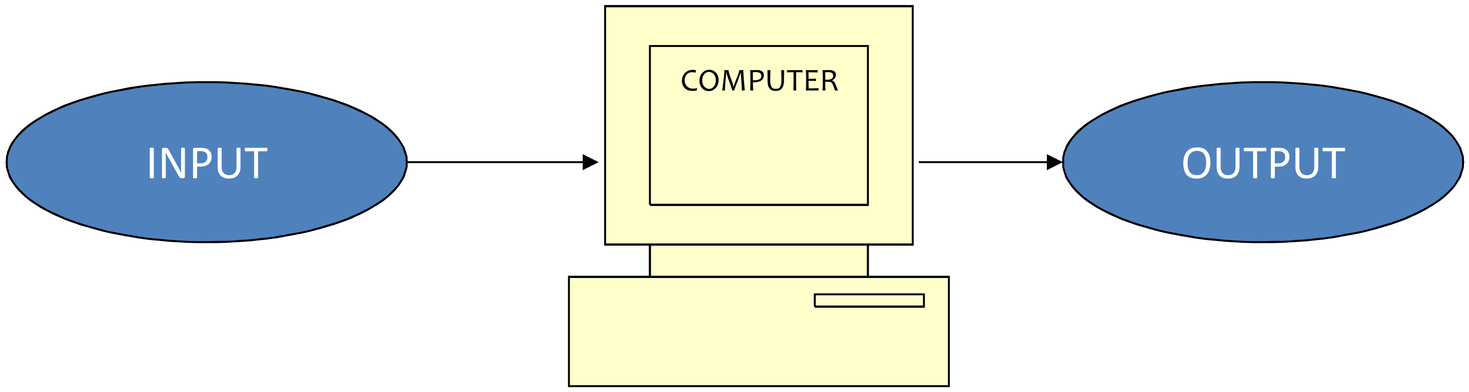
Lesson Objectives

- **To Understand the following concepts**
  - Introduction to programs
  - Introduction to Pseudocode
  - Usage of variables and operators
  - Introduction to control constructs
  - Introduction to Arrays and Linked List

IGATE
Speed.Agility.Imagination

What is a Program

- **A program is a set of instructions for a computer to perform a specific task.**
- **Programs can be written in one or more programming language**
- **Computers accept input, process it and generate output.**



Application, Program, and Software

- **Program**
  - A set of logically placed instruction to perform a task
- **Application program or application**
  - Any program designed to perform a specific functionality
- **Software**
  - A set of programs and associated documentation concerned with a specific operation stored electronically

Features

- **It is required that the programs should be:**
  - Readable (by others)
  - Maintainable
  - Modular
  - Reliable
  - Robust
  - Efficient
  - Easy to use
  - Flexible
  - Extendable
  - Reusable

Guidelines

- **“Coding Standards” have to be maintained.**
- **Coding Standards help to read others code, and maintain consistency. They entail standards regarding:**
  - naming conventions
  - indentation
  - commenting standards
  - use of global variables
  - modularity

Analysis Phase

- **Analyze the requirement**
  - Understand the problem
  - Gather correct information
    - Find proper data structure
    - Check any specific algorithm can be used

Design Phase

- **Design includes translation of the requirements into a logical structure that can be implemented in a programming language.**
- **The programmer designs the application flow/program using design tools such as**
  - **Flowchart**
    - A flow chart, or flow diagram, is a graphical representation of a process or system that details the sequencing of steps required to create output.
  - **Algorithms**
    - An algorithm is a set of instructions for solving a problem. It is a basic technique of how to do a specific task.
  - **Pseudo code**
    - A pseudocode is an algorithm expressed in a natural language rather than in a programming language.

Coding

- **Coding**
  - The logical tasks listed in the Pseudocode or flowchart are translated in a particular programming language.
  - The programmer checks the code's logic and syntax to ensure that they are correct.
  - Once the code is written, perform the following
    - Compile the code
      - Translating higher-level programming language code into machine language code
    - Review the code
      - Review the code using checklist to identify defects if any
    - Execute the code to verify the output

Review

- **To identify errors , either in the code or in other artifacts, self review is very important**
- **Self review**
  - Process of reading code line by line to check for:
    - Flaws or potential flaws
    - Consistency with the overall program design
    - The quality of comments
    - Adherence to coding standards
  - Is done by the developer with the help of checklist
  - If the evaluation is done by other people in the team ,it’s called as peer review

What is a Pseudocode?

- **A pseudocode is an algorithm expressed in a natural language rather than in a programming language.**
- **It is written in the design phase.**
- **It is an outline of a computer program, written in a format that can easily be converted into programming statements.**

Algorithm and Pseudocode

- Algorithm for finding GCD of two numbers

  Step 1 If *n* = 0, return the value of *m* as the answer and stop; otherwise, proceed to Step 2.

  Step 2 Divide *m* by *n* and assign the value of the remainder to *r*.

  Step 3 Assign the value of *n* to *m* and the value of *r* to *n*. Go to Step 1.

- Pseudocode for finding GCD of 2 numbers

```
//Computes gcd(m, n) by Euclid's algorithm
//Input: Two nonnegative, not-both-zero integers m and n
//Output: Greatest common divisor of m and n
while n = 0 do
r ←m mod n
m←n
n←r
return m
```

IGATE
Speed.Agility.Imagination

Why Pseudocode?

- Easy and Efficient Coding
- Increase the Quality of program
- Less cost activity
- Provides programmers a detailed template for the next step of writing instructions in a specific programming language.
- Pseudocode is used to bridge the gap between algorithms and programming languages
- If we develop the program logic by using the pseudocode, we can easily translate it in to code in any programming language.
- We can focus on the logic development without getting caught up in the syntax.

How to write Pseudocode?

- **All statements are written as sentence.**
- **Use unique variable names but there is no need to declare them before they are used.**
- **There is no universal "standard" Code for writing Pseudo Code.**

Best practices of writing pseudocode

- **There is no absolute standard for pseudocode, these best practices can be followed:**
  - Use simple English
  - Write each instruction on a separate line
  - Declare variables in the format of “DECLARE variablename as basictype”, if required
  - Use “initialize” keyword to initialize value to a variable.
  - Capitalize keywords
  - Follow indentation strictly
  - Group instructions into modules.
  - Always use terminators for loops and iteration like ENDLOOP, ENDIF
  - Provide only one entry and one exit point in a Pseudocode using BEGIN and END keyword.
  - Every program and module should have a header preceding it.
  - Module and variable names should be meaningful.
  - Follow all the programming best practices like readable, maintainable, etc.

Example of pseudocode

```
BEGIN
          DECLARE CONSTANT interest_rate =0.5
          INITIALIZE Amount = 0
          INITIALIZE Interest = 0
          INITIALIZE Ctr=0
          WHILE Ctr  <10
          DO
                    PRINT “Enter amount to find interest”
                    ACCEPT Amount
                    CALCULATE Interest = Amount * interest_rate
                    DISPLAY Interest
                    Ctr=Ctr+1
          END WHILE
END
```

What are variables, constants and Data Type?

- **Variables**
  - Variables are programmable placeholders for holding character, string, numeric and boolean values
  - They can be declared, initialized and processed
  - It can be global , local or static
- **Constants**
  - Constants are values that don't change throughout an application's lifetime
- **Data Type**
  - A Data Type defines how data is to be interpreted
  - A data type indicates what values can be taken and what operations can be performed
  - Data Type can be categorized as
    - Fundamental Data Types
    - Composite Data Type or User Defined Data Type

Fundamental Data Types

- **Fundamental data types are the data type provided as basic building blocks**
- **They are also known as primitives or basic data type, following table shows the basic data types**

Data Type	Description
Character	A character type (typically called "char") may contain a single letter, digit, punctuation mark, or control character. Some languages have two or more character types, for example a single-byte type for ASCII characters and a multi-byte type for Unicode characters
Integer	An *integer* data type can hold a whole number
Real	A *real* type stores rational number having fractional part
Boolean	A *boolean* type, typically denoted "bool" or "boolean", is a single-bit type that can be either "true" or "false".

IGATE
Speed.Agility.Imagination

Composite Data Types

- **A composite data type helps in grouping logically related data as one unit**
- **The data types derived/created from the fundamental data types are called Composite or User Defined data types**.

  Example:
  - Arrays
  - String
  - Records

```
RECORD Employee
    DECLARE Empno AS INTEGER
    DECLARE Emp_name AS STRING
    DECLARE Salary AS INTEGER
END RECORD
```

Introduction to Control Constructs

- **There are basically three control constructs used to write algorithms:**
  - **Sequence:** The instructions are executed in the sequence in which they appear, and the program does not skip or repeat any of the instructions
  - **Selection:** Selection implies that a choice will be made, which depends on the value of a condition specified by the programmer
  - **Repetition:** Repetition repeats a section of code while a certain condition holds true.

Guidelines for Conditional Statements

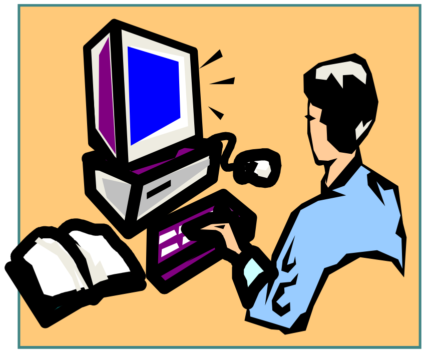
- **When to use the if statement**
  - When only one condition is being checked
- **When to use if else statement**
  - When more conditions are being checked and the subsequent conditions are related to the first condition
- **When to use multiple if statements**
  - When more conditions are being checked and the subsequent conditions are not related to the first condition
- **In case of multiple or nested IF conditions, implement the most common conditions at the beginning.**

Guidelines for Looping Statements

- **When to use a for loop**
  - When the iterative task is to be performed for <n> number of times
- **When to use a while loop**
  - When the question whether to continue the loop or not is asked at the beginning of the iterative task
- **When to use a do while loop**
  - When the question whether to continue the loop or not is asked at the end of the iterative task

Demo : Variables, Operators and Control Constructs

- **Refer the pseudo code available in the below listed files for understanding the usage of variables, operators and control constructs**
  - ArithmeticOperators
  - TernaryOperators
  - IF-ELSEIF-ELSE
  - LogicalOperators
  - DoUntil
  - WHILE-CYCLE-EXIT
  - ForLoop



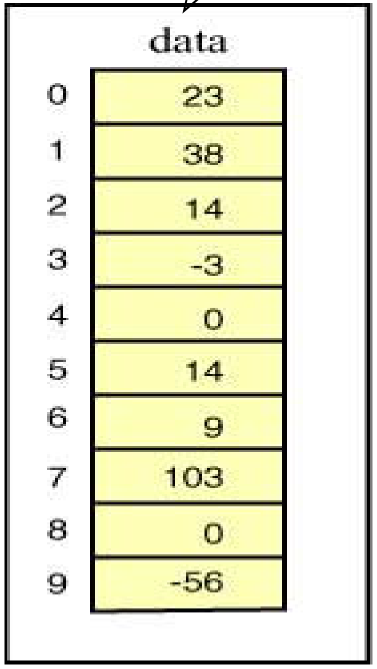
What is data structure

- A data structure is useful for organizing and storing data properly.
- Some data structures are linear and some are non linear.
- Linear data structure : If data structure organizes data in contiguous memory locations then it is called as linear data structure. E.g Arrays
- Non linear data structure : If data structure organizes data in non contiguous memory locations then it is called as non linear data structure. E.g Linked lists

Introduction to Arrays

- **Array**
  - It is a linear data structure that is used to store a list of values.
  - It is made out of a contiguous block of memory that is divided into a number of "slots."
  - Each slot holds a value, and all the values are of the same type, addressable by index or subscript, usually starting with 0
  - Are useful when a defined set of data has to be processed systematically
  - Make a program handle large amount of data without having to write unnecessary code

DECLARE array[10] AS INTEGER ARRAY
can be represented as



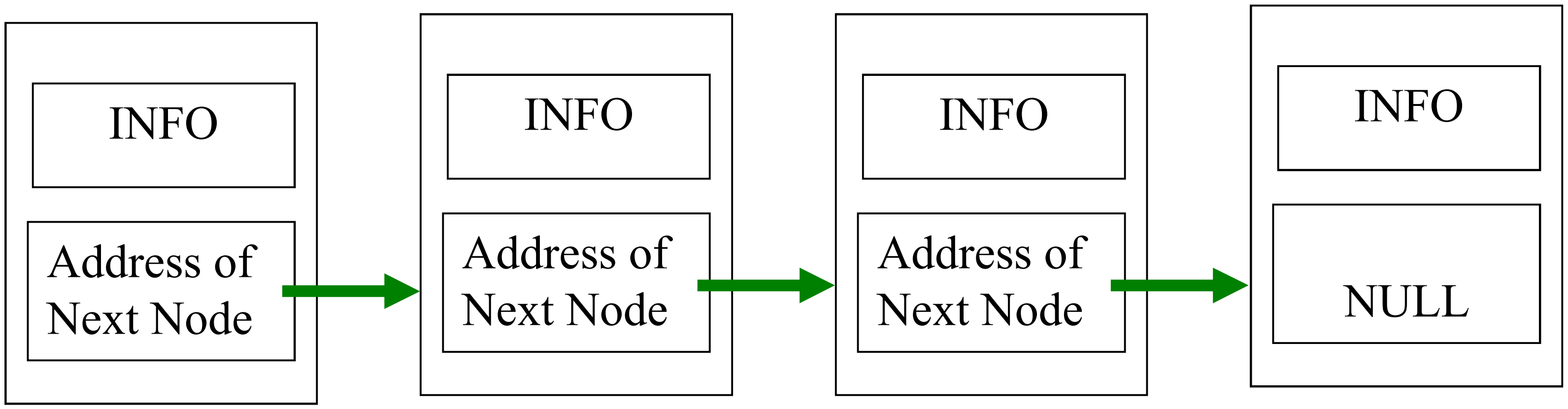
Example of Arrays

- **Find out the maximum number among 10 numbers**

```
BEGIN
    DECLARE numbers[10] AS INTEGER ARRAY
    DECLARE max AS INTEGER
    INITIALIZE max TO 0
    FOR index=1 TO 10
          ACCEPT numbers[index]
    END FOR
    max=numbers[0]
    FOR index=1 TO 10
        IF numbers[index] > max THEN
            max=numbers[index]
        END IF
    END FOR
    PRINT max
END
```

Non Linear data structure – Linked list

- **Linked List is a linked list of structures (called nodes) for which memory is allotted dynamically.**
- **There are two kinds of linked lists. They are:**
  - Singly Linked Lists
  - Doubly Linked Lists

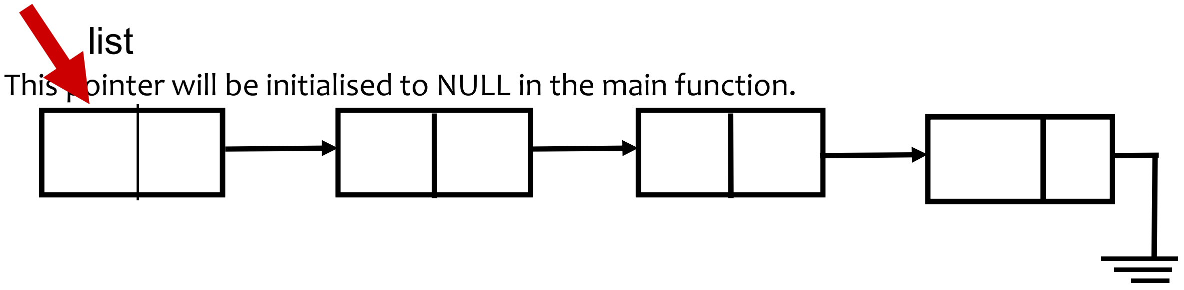


Salient Features

- **Singly Linked List:**
  - For a Singly Linked List, each node points only to the next node.
  - It possesses dynamic data structure, i.e., it can grow or shrink in size during execution of a program.
  - It uses the memory that is needed for the list.
  - It provides flexibility to rearrange the nodes efficiently.
  - It is easier to insert or delete items by rearranging the links.

Process Steps

3. Check the start of list pointer's declaration.
   - The list needs a pointer that will point to the beginning of the list.
   - This pointer will be initialised to NULL in the main function.



Insert first node

- **Defining the Insert function:**
  - This function will take the “list pointer” (indicate Starting of the list) and “address of the node” to be inserted as a parameter. If the list is empty, then the new node becomes the first node:

The pseudo code to describe the algorithms uses these terms

list = pointer to the first list item (head node).

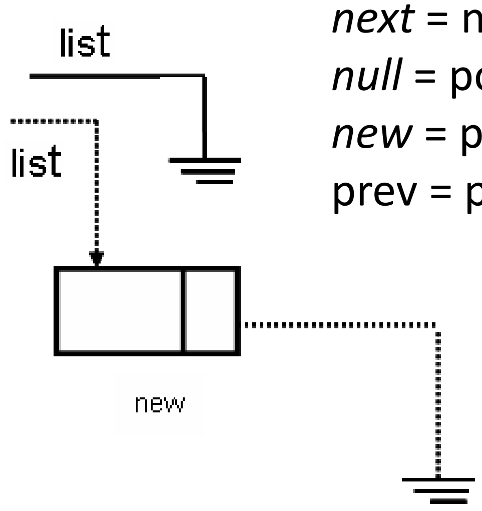
info = node data

next = node link

null = pointer that is not an actual address

new = pointer to a node

prev = pointer to previous node

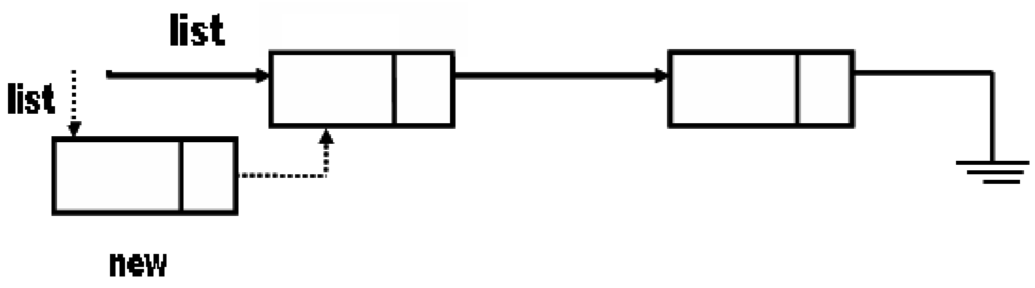


Different Scenarios

- **Insert function should take care of the following:**
  1. Inserting a Node at the Beginning of the list
  2. Inserting a Node in the Middle of the list
  3. Inserting a Node at the End of the list

Different Scenarios

1. **Inserting a Node at the Beginning of the list**



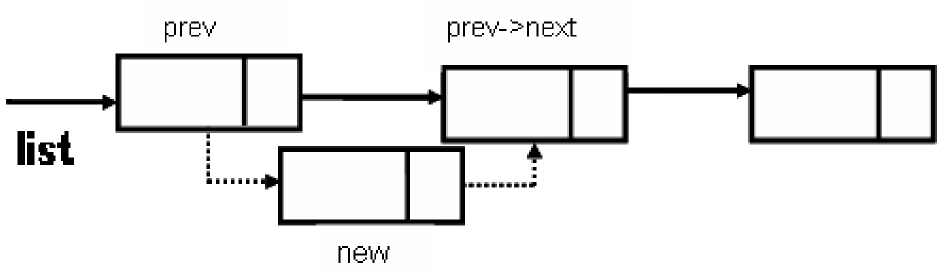
The new node has to point to the existing first element of the list. This is done by the following statement:

new -> next=list;

The list pointer must point to the new node:

list=new;

2.



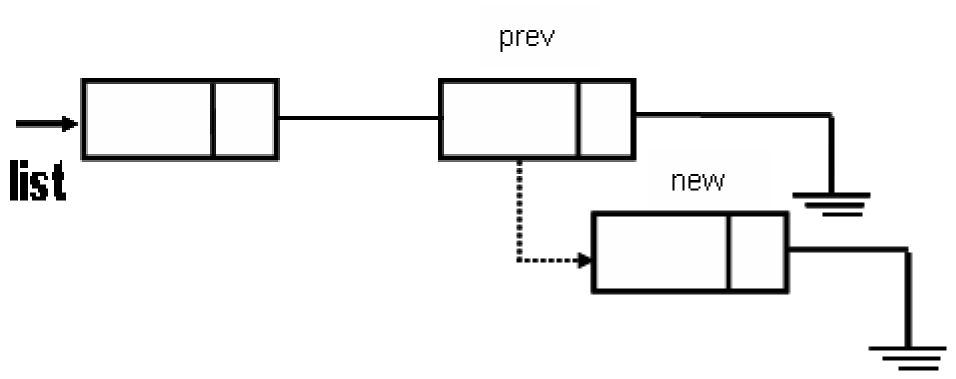
The new node has to be inserted after the node pointed by prev node, then previous node should point to new, and new will point to next of prev.

new ->next = prev-> next;

prev -> next = new;

Different Scenarios

3. **Inserting a Node at the End of the list**



The last node should point to new node, and new node should point null to become last node.

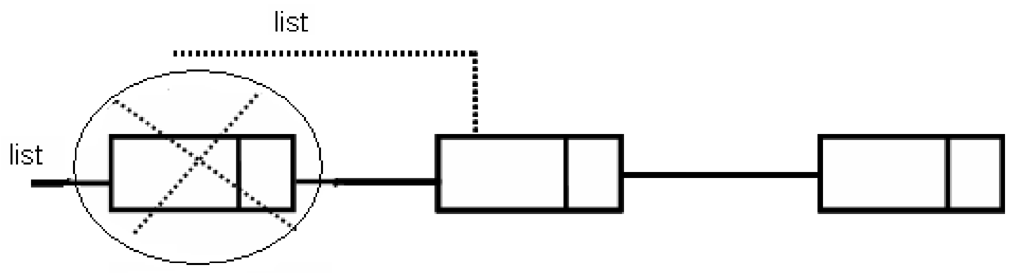
```
new -> next = null
which is equivalent to
new-> next = prev-> next;
then
prev-> next = new;
```

Different Scenario

- **To delete a node, the links have to be reformulated to exclude the deleted node. The memory allocated for the deleted node must also be freed.**
- **The node to be freed can exhibit the following traits:**
  - It may not be existing in the list.
  - It may be the first node (in which case the list pointer must be reinitialised).
  - It may be any other node or the list may be empty.

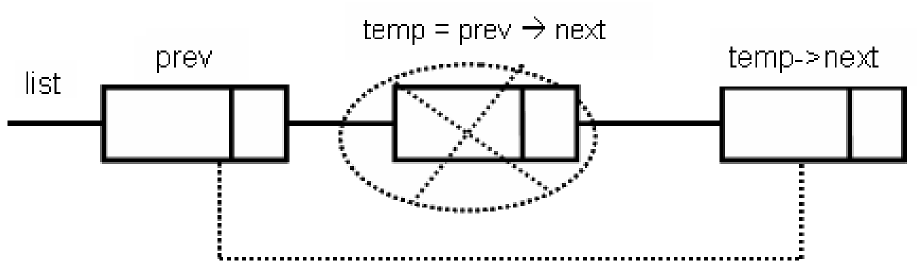
Different Scenario

- **To delete the first node:**



```
temp=list;
/* where temp is
  defined as node*/
   list = list->next;
   destroy the temp node;
```

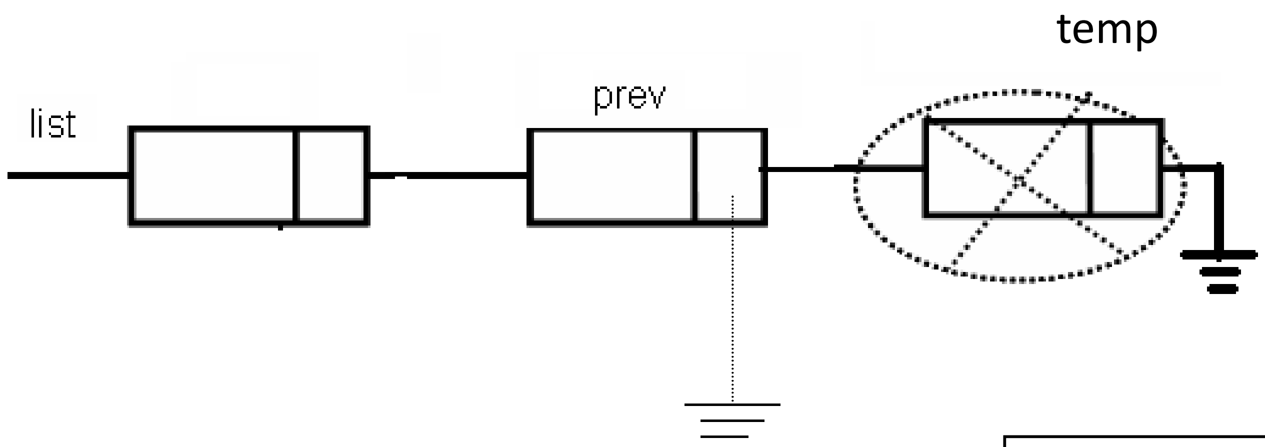
-



```
temp=prev->next;
/*   prev   is   the   node
  returned by search */
prev->next=temp->next;
destroy the temp node;
```

Different Scenario

- **To delete the node from the end:**



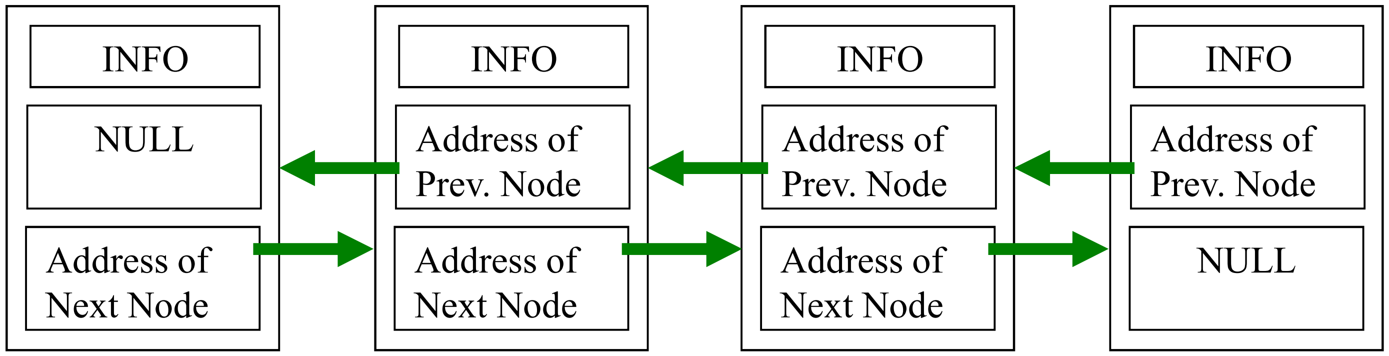
```
temp=prev->next;
/*   prev   is   the   node
  returned by search */
prev->next=temp->next;
destroy the temp node;
```

Explanation

- **Limitations of Singly Linked List:**
  - It results in loss of information, if any one of the link is corrupted.
  - It can traverse only in the forward direction.

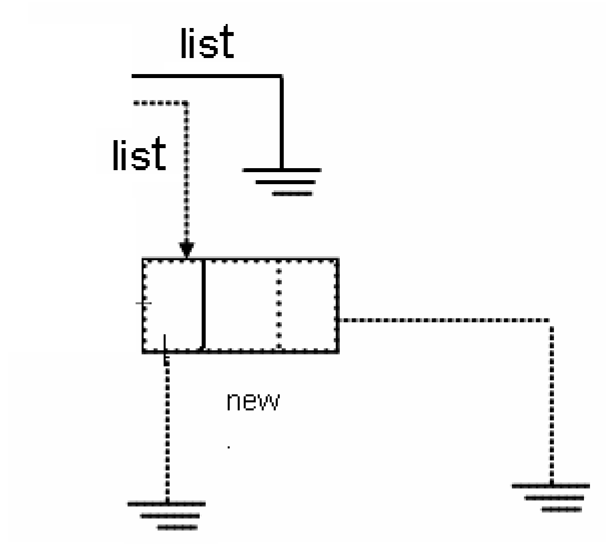
Salient Features

- **A doubly linked list possesses the following traits:**
  - It possesses two pointers – one points to the next node and the other points to previous node.
  - It can be traversed in both directions.
  - It is easy to restore the information even if any one of the link is corrupted.



Explanation

- **Defining the Insert function:**
  - This function will take the “list pointer” (start of the list) and “address of the node” to be inserted as a parameter. If the list is empty, then the new node becomes the first node as shown below:

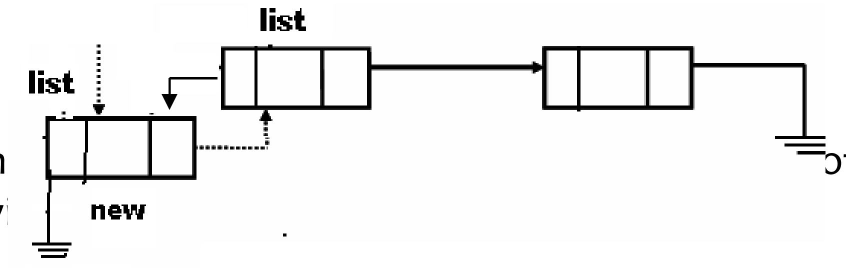


Different Scenario

- **Insert function should take care of the following:**
  1. Inserting a Node at the Beginning of the list
  2. Inserting a Node in the Middle of the list
  3. Inserting a Node at the End of the list

Different Scenario

1. **Inserting a Node at the Beginning of the list**



- The new n of the list. This is done by the followi
- The list pointer must point to the new node:

```
new -> next=list;
     new->prev=list->prev
```

```
list->prev=new
     list=new;
```

Comparison : Arrays – Linked

Operation	Arrays	Linked List
Accessing Elements	Random Access	Sequential access. Time required to access element is O(n)
Memory Allocation	Memory allocation is contiguous and allocation is done at compile time. Hence called as Static allocation	Elements are stored at any available location but pointer ti first element and next element is stored Memory is allocated dynamically at the time of execution. Hence called as static allocaion
Insertion/deletion	Insertion/deletion at end is easily done. Insertion/ deletion in between requires shifting of rest of the elements, hence time consuming	Insertion nd deletion at any place is easy, because only appropriate pointer assignment is required.
Space requirements	Space may get wasted	Space doesn't get wasted. But extra space is required for storing addresses.

IGATE Sensitive

Lab

- **Min 3 Logic building programs from Question Bank**

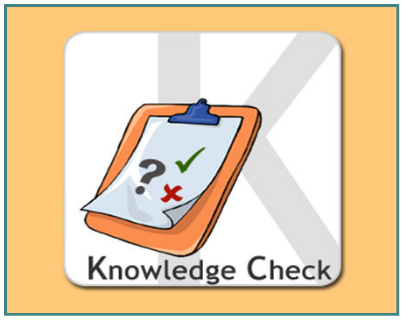
Summary

- **In this lesson, you have learnt about:**
  - Introduction to programs
  - What is a program?
  - What is an application?
  - Industry Vs College Programs
  - Types of projects
  - SDLC process of waterfall model
  - Introduction to Algorithm and Pseudocode
  - Usage of variables, datatypes and constants, Control structure

IGATE
Speed.Agility.Imagination

Review Question

- Question 1: When the question whether to continue the loop or not is asked at the beginning of the iterative task the ------ loop is better to use.
- A. For Loop
- B. While loop
- C. do-while loop
- D. Any of the above



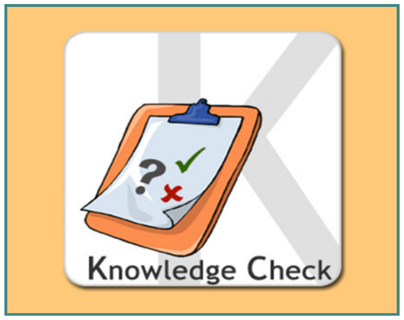
- **Question 2: A task which gets done in the specified time with desired quality defines ------**
  - A. Maintainable
  - B. Efficient
  - C. Robust
  - D. Readable

IGATE
Speed.Agility.Imagination

Review Question

- **Question 3: There is no absolute standard for pseudocode**
  - A. True
  - B. False

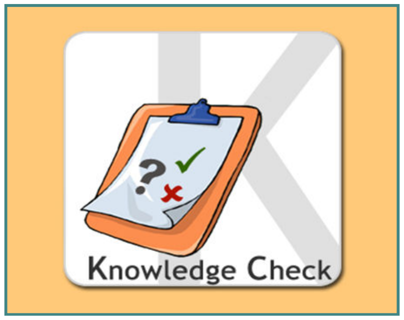
- **Question 4: Identify guidelines for using variables**
  - A. Assigning values to variables just before values are used
  - B. Prefer local variables over global ones
  - C. Initialize variables used in looping constructs at the top of the code block
  - D. Maximize the lifetime of local variables



IGATE
Speed.Agility.Imagination

Review Question

- **Question 5: Identify the hazardous aspects of using arrays**
  - A. Empty arrays always raise errors
  - B. Errors can occur because of inappropriately defined bounds
  - C. Its easy to misuse indexes
  - D. Null values in place of empty arrays are difficult to handle

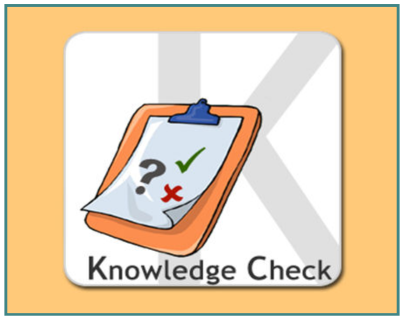


IGATE
Speed.Agility.Imagination

Review Question

- **Question 6: How does the following code breach best practice guidelines**

```
myarray[10]= new Array{1,1,2,3,5,8,13,21,34,55};
index=0;
while(index<10)
index=index+1;
//dosomething
```

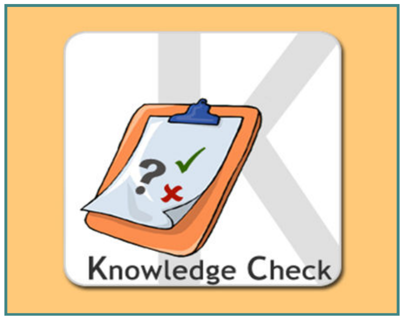


  - A. The array should have a clearer name
  - B. The index is never incremented
  - C. The index Is zero based so the loop passes beyond the last element

IGATE Speed.Agility.Imagination

Review Question

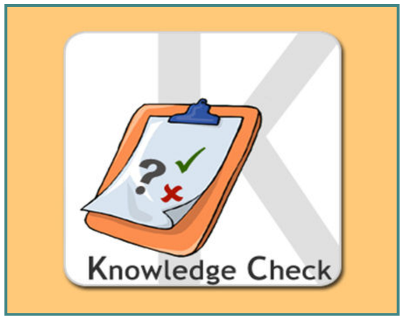
- **Question 7: Match the looping structure to their definitions**



1. For	A. Condition controlled for executing choice once
2. IF	B. Condition controlled for executing choice multiple times
3. While	C. Count controlled

Review Question

- **Question 8: You need to create a loop whose exit condition depends solely on the maximum number of employees being reached when you increment the number of employees by one. Which construct should you choose to ensure greatest clarity**
  - A. For
  - B. DO UNTIL
  - C. while



IGATE
Speed.Agility.Imagination