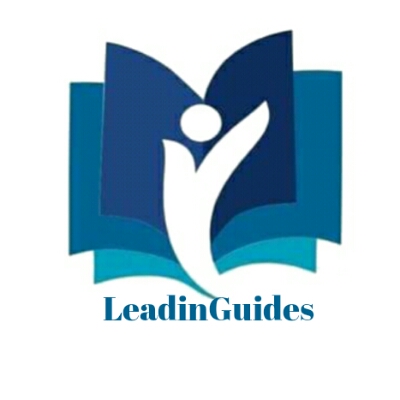
**FIRST TERM – 20\_\_ \_\_/20\_\_ \_\_ ACADEMIC SESSION**

SCHOOL: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Editable School badge



LESSON PLAN FOR THE FIRST WEEK ENDING \_\_\_ - FRIDAY, \_\_\_\_\_\_\_\_\_\_\_, 20\_\_ \_\_\_

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Term | 3 | Week | 2 | Date |  | Teacher |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Class | SSS 2 | Age | 15–17 yrs | Period |  | Duration | 60minutes |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Subject | Computer Science | | Standards | SSCE, WAEC, NECO, IB, IGCSE |
| Topic | Algorithms and Flowcharts | | Sub-topic | Meaning & Differences |
| Learning Objectives | Cognitive | 1. List the stages in System/Program Development Life Cycle (SDLC/PDLC). 2. Define algorithm. 3. Differentiate between at least 3 ways of expressing algorithms 4. Define key programming terms. | | |
| Affective | 1. Demonstrate critical/logical thinking & systematic problem-solving skills. | | |
| Psychomotor | 1. Participate in hands-on group activities to simulate and design algorithmic processes. | | |
| Prior Knowledge | Students understand the concept of problems and the meaning of computer programs and programming. | | | |
| Resources & Teaching Aids | * Projector/Whiteboard, * Chart of Flowchart Symbols, * Sample requirement specification documents. * Alarm App and Classroom Robot design examples * Past WAEC/IGCSE/IB exam questions | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method of Teaching | Eclectic – Interactive demonstration, Group-based simulation, Guided discussion, Question-and-Answer, Hands-on design tasks | | Differentiation | * Visual learners: flowcharts and structure diagrams. * Logical learners: pseudocode and abstraction. * Kinesthetic learners: set induction simulation game |
| Teacher’s Activities | | Learners’ Activities | | |
| 1. Explains concepts, 2. Moderates group work, 3. Provides examples, 4. Assesses performance. | | 1. Participate in simulations, 2. Group analysis, 3. Note-taking, 4. Practical problem-solving tasks. | | |
| Keywords | Algorithm, SDLC, PDLC, Abstraction, Decomposition, Pseudocode, Flowchart, Structure Diagram | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Presentation | | | |
| Time | Content & Teacher’s Role | Learner’s Role | Resource |
| 5minutes | **Introduction: Set Induction**   * Conduct the “Commander & Soldier” classroom activity to simulate instruction-following. * Highlight communication clarity as essential in programming. * Prompt students to reflect on real exam questions from WAEC/IGCSE/IB | * Participate in role-play simulation with group members (commander, soldier, observer). * Follow and document commands during the activity. * Reflect on outcomes and relate experience to programming logic. * Attempt select related past questions. | * Desks/objects for obstacles, * Printed WAEC/IGCSE/IB past questions, * Observer note sheets |
| 25 minutes | **SDLC & Analysis Phase**   * Introduce SDLC and PDLC. * Explain each phase: Analysis, Design, Implementation, Testing, Maintenance. * Emphasize the importance of requirement specifications. * Discuss abstraction and decomposition * **AFL (5 minutes)**: Short multiple-choice quiz to assess comprehension (Worksheet Ex2) | * Group discussion * Review real examples (Alarm App, Robot App) | * SDLC Phase Diagram - A poster/digital slide showing the 5 stages (Analysis, Design, Coding, Testing, Maintenance) with simple icons. * Sample Requirements Specification Document - Simplified 1-page examples (e.g., alarm app, robot app) to distribute or project. |
| 25 minutes | **Algorithm & Representation**   * Define algorithm and explain its key characteristics * Teach students about natural language, flowchart, pseudocode, and structure diagram * Use Alarm App scenario for practical examples * **AFL (5 minutes):** Complete matching tables and diagram-based tasks (e.g., structure diagrams, pseudocode exercises) | * Group decomposition of Alarm App tasks * Identify repeated tasks for modularization | * Alarm App Scenario Description * Flowchart Diagram Samples * Pseudocode Examples * Board/Markers/Chalk |
| 5 minutes | Conclusion   * Recap main concepts * Summarize lesson content for student notes * Provide feedback, mark participation, and support struggling learners | * Listen attentively to summary of the lesson * Copy the teacher’s final notes into their notebooks * Ask questions or seek clarification where needed | Student notebooks, teacher’s summary notes, classwork responses |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| NOTES | | | | | | | | | | | | | | |
| **Program (System) Development Life Cycle** | | | | | | | | | | | | | | |
| In computers, most projects (problem-solving exercises) are in cycles. When we say something is in a circle, we mean that | | | | | | | | | | | | | | |
| it has a defined path that it follows from start to finish repeatedly. This enables professionals to easily replicate solutions, | | | | | | | | | | | | | | |
| learn from previous experiences, and improve upon subsequent ones. | | | | | | | | | | | | | | |
| The standard (well-defined) paths that software developers follow to develop and improve the software is called | | | | | | | | | | | | | | |
| System Development Life Cycle (SDLC) or Program Development Life Cycle (PDLC). While the system. | | | | | | | | | | | | | | |
| development life cycle is generic, the program development life cycle applies specifically to software development. | | | | | | | | | | | | | | |
| **Phases of SDLC/PDLC** | | | | | | | | | | | | | | |
| There are six phases in a System or Program Development Life Cycle. These include: | | | | | | | | | | | | | | |
| 1. Planning | | | | | | | | | | | | | | |
| 1. Analysis & Feasibility Study | | | | | | | | | | | | | | |
| 1. Design | | | | | | | | | | | | | | |
| 1. Development (Programming or Coding) | | | | | | | | | | | | | | |
| 1. Implementation or testing | | | | | | | | | | | | | | |
| 1. Maintenance | | | | | | | | | | | | | | |
| **PDLC Phase: Analysis** | | | | | | | | | | | | | | |
| The analysis is the first stage of SDLC in which **needs** are assessed and business questions are translated into engineering | | | | | | | | | | | | | | |
| problems by considering factors such as cost (how much will this cost?), performance, functionalities (what features will | | | | | | | | | | | | | | |
| this have?) and risks (what are the dangers involved in this?). | | | | | | | | | | | | | | |
| In this phase, all the people involved in the project (stakeholders including developers, users and owners) will write out the | | | | | | | | | | | | | | |
| needs in clear terms so anyone working on the solution will understand what is needed. | | | | | | | | | | | | | | |
| The clearly documented needs (what is expected) of the software are called **requirements specifications** of the program. | | | | | | | | | | | | | | |
| A real-life requirement specification is a comprehensive document. It is usually arranged under different headings starting from introduction, purpose, scope, functional and non-functional requirements, etc. | | | | | | | | | | | | | | |
| There are two tools in the analysis phase to ensure that requirements specification is truly specific. These are **abstraction** and **decomposition**. | | | | | | | | | | | | | | |
| **Abstraction** | | | | | | | | | | | | | | |
| Abstraction is the process of removing unnecessary details from a problem or solution. We may:  - Abstract a problem – this is called problem abstraction.  - Lengthy or multiple procedures – this is called procedural abstraction.  - Bulky solution – this is called solution abstraction.  - Overwhelming (too much) data – this is called data abstraction.  - Too many functions or features – this is called functional abstraction. | | | | | | | | | | | | | | |
| **Benefits of Abstraction** | | | | | | | | | | | | | | |
| The benefits of abstraction include:   * Reduced development time because factors that can detract from the program can be ignored. * It makes the program more likely to solve the problem because unnecessary aspects will not detract from the main purpose of the program. * Reduces complexity of programming code because therefore can run on lower spec computers. | | | | | | | | | | | | | | |
| **How to Abstract** | | | | | | | | | | | | | | |
| To abstract a requirements specification, write out only what the system will do. You may also summarize the performance and functional requirements, each in one or two sentences. | | | | | | | | | | | | | | |
| **Example: Abstracted Requirements Specification for A Simple Alarm App** | | | | | | | | | | | | | | |
| The app shall allow users to set a single alarm with customizable settings, including alarm time, sound, snooze time, label, and volume. The notification for the alarm shall include the label and time for the alarm. | | | | | | | | | | | | | | |
| **Decomposition** | | | | | | | | | | | | | | |
| Decomposition is the process of breaking down a problem or solution into smaller, more manageable parts. Any problem that uses a computer system for its solution needs to be decomposed into its component parts.  The benefit of decomposition is that it makes complex problems easier to solve and manage. | | | | | | | | | | | | | | |
| **Computer System, Automation & Subsystem** | | | | | | | | | | | | | | |
| A computer system is any system that is made up of software, data, hardware, communications and a person(s).  Creating or maintaining a computer system for business problems (i.e., automation) is a problem that requires a systematic approach.  For complex computer systems, the first step is to divide them into smaller parts. The small parts which combine to form a complex system are called sub-systems.  Sometimes, a system may be so large or complex that we have to break it down into multiple downlines of sub-systems. That is, we break the initial complex system into the major sub-systems. Then, we break the major sub-systems into a set of further sub-systems until each sub-system is simple enough to perform only one task. | | | | | | | | | | | | | | |
| **Benefits of Decomposition** | | | | | | | | | | | | | | |
| The aim of decomposing a problem is to make the problem easier to solve. Decomposing a system also makes the system easier to manage and maintain. | | | | | | | | | | | | | | |
| **How to Decompose a Problem or System** | | | | | | | | | | | | | | |
| The process of breaking a complex system down into smaller sub-systems is called **stepwise refinement** or **top-down design**.  This is one of the standard approaches that developers employ in solving any problem. They do this by breaking down abstracted requirements specifications into the component parts of a computer system. | | | | | | | | | | | | | | |
| The component parts of any computer system are:  ****inputs****– the data used by the system that needs to be entered while the system is active  ****processes****– the tasks that need to be performed using the input data and any other previously stored data  ****outputs****– information that needs to be displayed or printed for the users of the system  ****storage****– data that needs to be stored in files on an appropriate medium for use in the future.  To decompose a problem is to identify the inputs, the processes, the outputs and the storage. | | | | | | | | | | | | | | |
| **Example: Decomposed Alarm App** | | | | | | | | | | | | | | |
| * **inputs** – time to set the alarm, remove a previously set alarm time, switch an alarm off, and press the snooze button. * **processes** – continuously checking if the current time matches an alarm time that has been set, storing and removing alarm times, and management of snooze. * **outputs** – continuous sound/tune (at alarm time or after snooze time expired). * **storage** – time(s) for alarms set. | | | | | | | | | | | | | | |
| **Composition** | | | | | | | | | | | | | | |
| Composition is the process of combining sub-systems into compound or complex systems. Initially, you decompose to make solving the problem easier. After solving the problem, you may need to recombine the different parts to form a complete solution for the complex problem you initially set out to solve. | | | | | | | | | | | | | | |
| **Subroutines, Functions, Procedures & Sub-programs** | | | | | | | | | | | | | | |
| After decomposing a problem or system into sub-systems that perform one task, the instructions or program for each of such tasks may not be the complete software on their own. Instead, each is a part of the entire software that performs one particular task of the sub-system.  This kind of instruction is called **routine or sub-routines**. Thus, *we can define routines or subroutines as a sequence of program instructions that perform a specific task, packaged as a unit*.  This unit can be used in programs wherever that particular task needs to be performed. | | | | | | | | | | | | | | |
| **Types of routines or subroutines** | | | | | | | | | | | | | | |
| There are different terms that identify subroutines based on how developers use them. These include:   * **Library routine** – A standard subroutine that is available for immediate use. * **Procedure** – A subroutine that may not return a value. It only controls the flow. * **Function** – A subroutine that can be used assigned, reassigned, called and recalled many times in a program with a different set of data. | | | | | | | | | | | | | | |
| **Algorithm** | | | | | | | | | | | | | | |
| An algorithm is a list of clear steps to solve a problem. A good algorithm is simple, clear, and has an end. | | | | | | | | | | | | | | |
| Features of a Good Algorithm | | | | | | | | | | | | | | |
| A good algorithm should be:   * simple * clear with no ambiguity * a unique solution to the problem * finite in the number of steps to arrive at a solution * able to handle unexpected situations. | | | | | | | | | | | | | | |
| **Ways of Representing Algorithm** | | | | | | | | | | | | | | |
| The various ways of representing or writing algorithms are:   * Natural Language * Flowchart * Pseudocode * Structure Diagrams | | | | | | | | | | | | | | |
| Flowchart | | | | | | | | | | | | | | |
| A flowchart is a diagram made of two or more different standard symbols (shapes), lines and arrows which represent an algorithm, the steps required to complete a task and the order that they are to be performed. | | | | | | | | | | | | | | |
| **Benefits of Flowcharts** | | | | | | | | | | | | | | |
| Flowcharts offer several benefits over natural language for representing a process or algorithm, including:   * **Visual representation**: Flowcharts provide a visual representation of the process, which makes it easier to understand the steps involved and the overall flow of the process. * **Clear and concise**: Flowcharts can convey complex ideas in a clear and concise manner, making them easier to comprehend than long blocks of text. * **Standardized notation**: Flowchart symbols and notation are standardized, which makes it easier for different people to understand and interpret them. * **Error detection**: Flowcharts can help identify errors and inconsistencies in a process, making it easier to correct them before they cause problems. * **Simplifies complexity**: Flowcharts can help simplify complex processes by breaking them down into smaller, more manageable steps. * **Improves communication**: Flowcharts can improve communication and collaboration among team members, particularly when working on complex projects. | | | | | | | | | | | | | | |
| **Pseudocode** | | | | | | | | | | | | | | |
| Pseudocode is a written form of an algorithm using keywords, operators and names for data and routines that are similar to those in a high-level programming language but without following the syntax (rules) of any particular programming language. | | | | | | | | | | | | | | |
| **Structure Diagram** | | | | | | | | | | | | | | |
| A Structure Diagram is a diagram made up of mostly rectangles or squares connected by lines in hierarchical order representing the breakdown of a system into component sub-systems. | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | |
| Summative Assessment | | | | | | | | | | | | | | |
| Classwork (Question or References) | | | | | | | Homework (Questions or Referenc | | | | | | | |
| Click here to check the lesson notes for classwork exercises to include here. | | | | | | | Click here to check the lesson notes for homework exercises to include here. | | | | | | | |
| Pass Mark |  | % Pass |  | % Fail |  | Highest Score | | |  | | Lowest Score |  | | |
| Supervisor’s  Remark |  | | | | | | | | | | | | | |
| Name | |  | | | | | Date | |  | | | Sign |  |