# Skill class:

1. Attributes:

`id`: A unique identifier for each skill, generated automatically.

`name`: A string to store the name of the skill (e.g., "Testing," "Designing").

`proficiency`: A floating point number to represent the proficiency level associated with the skill.

`next`: A pointer to the next skill in a linked list.

2. Constructor: The class has a constructor that initializes a skill object when it's created. It assigns a unique `id` based on the order of creation, sets the skill's `name` and `proficiency` as specified when creating the object, and initializes the `next` pointer as `nullptr`.

3. Linked List: The class maintains a linked list of `Skill` objects through the `next` pointer, allowing for the creation and management of multiple skills in a structured manner.

4. Unique ID Generation: It provides a mechanism to generate unique IDs (`id`) for each skill. The `next\_id` static variable is used for this purpose, ensuring that each skill is assigned a unique identifier based on the order of creation.

This code, therefore, functions as a foundation for creating, representing, and managing various skills within the program.

# Task class:

1. Attributes:

`id`: A unique identifier assigned to each task, starting from 1 and incremented for each new task.

`duration`: An integer value that indicates the estimated duration or effort required to complete the task.

`skill`: A pointer to a `Skill` object, representing the specific skill or expertise required for the task.

`next`: A pointer to the next task in a linked list of tasks.

`dependency1` and `dependency2`: Integer values that specify dependencies on other tasks. Each task can potentially have up to two dependencies on other tasks.

2. Constructors: The class provides a constructor for creating `Task` objects. The Constructor initializes a with other attributes such as `earlyStart`, `lateStart`, `earlyFinish`, `lateFinish`, and `slack` with default values of 0.

3. Unique ID Generation: Similar to the `Skill` class, the `Task` class uses the `next\_id` static variable to generate unique IDs for each task, ensuring that tasks are uniquely identified based on their order of creation.

# Resource class:

1. Attributes:

`id`: A unique identifier assigned to each resource, starting from 1 and incremented for each new resource.

`availability`: An integer value that indicates the availability or capacity of the resource for taking on tasks. It typically represents the number of hours or units of work the resource can contribute.

`skills`: A pointer to a linked list of `Skill` objects. This list represents the specific skills possessed by the resource.

`numSkills`: An integer that keeps track of the number of skills associated with the resource.

`next`: A pointer to the next resource in a linked list of resources.

2. Resource Skills: Resources are associated with skills required to complete tasks. The `skills` attribute is a pointer to a linked list of `Skill` objects that represent the expertise areas of the resource. The `numSkills` attribute keeps track of the total number of skills linked to the resource.

3. Unique ID Generation: Similar to the `Task` and `Skill` classes, the `Resource` class uses the `next\_id` static variable to generate unique IDs for each resource, ensuring that resources are uniquely identified based on their order of creation.

4. Resource Availability: The `availability` attribute represents the resource's capacity or availability for taking on tasks within a project. It could signify the number of hours per day or any other relevant unit of measurement.

# Project Class:

The `Project` class is designed to manage projects, tasks, and resources. Here's an explanation of each function's working and functionality:

1. Project() Constructor:

Initializes the project with empty task and resource lists, and sets the number of tasks and resources to 0.

2. addResources():

Allows the user to input and add resources to the project.

It takes user inputs for the number of resources and their initial availability.

For each resource, it allows the user to assign skills.

Resources are added to the linked list of resources.

3. addTask():

Allows the user to input and add tasks to the project.

It takes user inputs for the number of tasks, task durations, and their dependencies.

Tasks are added to the linked list of tasks, where each task can have up to two dependencies.

4. setTaskDuration(int newDuration):

Sets the same duration for all tasks in the project.

5. set\_nth\_TaskDuration(int taskId, int newDuration):

Sets the duration for a specific task (specified by `taskId`) in the project.

6. printTaskDependencyList():

Displays the dependencies of each task. It lists which tasks depend on other tasks.

7. calculateBasicSchedule():

Calculates the early start, early finish, late start, late finish, and slack for each task using the Critical Path Method (CPM).

Displays a table with these values for each task.

8. printCriticalTasks():

Calls `calculateBasicSchedule()` to calculate and display the critical tasks.

Identifies and displays the tasks that are part of the critical path (slack = 0) along with their start and finish times.

Calculates and displays the project's completion time.

9. Display():

Displays the tasks along with their durations and dependencies in a tabular format.

Note:

The `Project` class serves as a central component to manage a project's tasks, resources, and dependencies. It allows for the addition of tasks and resources, setting task durations, and identifying critical tasks using CPM. The `printCriticalTasks()` function is particularly useful for determining which tasks are critical to the project's completion.

# Main Function:

The `main` function initializes a `Project` object and enters a loop that displays a menu for users to choose from various options. It keeps running until the user chooses to exit.

## Menu Options:

The program presents the following menu options:

Add Resources (Option 1): Allows users to add resources to the project. Users can specify the number of resources and their initial availability, as well as assign skills to each resource.

Add Tasks (Option 2): Enables users to add tasks to the project. Users can specify the number of tasks, their duration, and dependencies on other tasks.

Set Task Duration (Option 3): Allows users to set a new duration for all tasks in the project.

Set nth Task Duration (Option 4): Allows users to set a new duration for a specific task by providing the task ID.

Print Task Dependency List (Option 5): Displays the list of task dependencies, indicating which tasks depend on which others.

Calculate Basic Schedule (Option 6): Calculates and displays the basic schedule for the project, including early start, early finish, late start, late finish, and slack for each task.

Print Critical Tasks (Option 7): Identifies and displays critical tasks with a slack of 0, calculating the project completion time.

Display (Option 8): Displays a summary of the project, showing task IDs, durations, and their dependencies.

Exit (Option 9): Exits the program.

## User Input and Processing:

The program reads the user's choice, performs the corresponding task based on the menu selection, and then returns to the menu to allow further interactions.

It includes error checking for invalid input, such as negative durations or invalid task IDs.

## Project Management:

The `Project` class, which is used within the program, provides functionalities for adding resources and tasks, setting task durations, managing dependencies, calculating schedules, identifying critical tasks, and displaying project information.

## Loop and Exit:

The program continues to run in a loop until the user chooses to exit by selecting option 9.

Overall, this C++ program is designed to help manage a project by providing a userfriendly interface to add resources, define tasks, set durations, manage dependencies, and perform scheduling and critical path analysis. Users can interact with the program to make changes to the project and retrieve important project management information.  
  
References:

<https://asana.com/resources/critical-path-method>

<https://www.projectmanager.com/guides/critical-path-method>

https://www.tutorialspoint.com/cpp\_standard\_library/iomanip.htm