1. Initialization and Main Menu

Code Explanation:

```
main proc

2Again:

3    call clrscr

4    mWrite<" 1-Quick Play",0dh,0ah," 2-Instruction",0dh,0ah,"
3-Setting",0dh,0ah>

5    mWrite<" 4-High Score",0dh,0ah, " 5-Quit",0dh,0ah>

6    mWrite<"Enter Choice:",0>

7    mov eax,0

8    call readdec
```

- What Happens: The program clears the screen and displays a menu with options for the user. It waits for the user to enter a choice.
- Key Features:
  - `clrscr`: Clears the console screen.
  - `mWrite`: Displays text on the screen.
  - `readdec`: Reads a decimal number from the user.

Potential Questions:

1. What is the purpose of the `clrscr` function?
   - Answer: The `clrscr` function is used to clear the console screen, providing a fresh display for the menu options.

2. How does the program handle user input?
    - Answer: The program uses the `readdec` function to read a decimal number input from the user, which corresponds to their choice from the menu.
3. What happens if the user enters an invalid choice?
    - Answer: If an invalid choice is entered, the program displays an error message and prompts the user to enter a valid option again.

---

2. Quick Play Procedure

Code Explanation:

```
Quick_play PROC

    call clrscr

    call level1

    call clrscr

    call level2

    call clrscr

    call level3

    ret

Quick_play endp
```

- What Happens: This procedure sequentially calls the three levels of the game, clearing the screen between each level.
- Key Features: It allows the player to progress through the game levels in order.

Potential Questions:

1. How does the `Quick_play` procedure manage the flow of the game?
    - Answer: The `Quick_play` procedure calls each level in sequence, allowing the player to play through all levels one after another. The screen is cleared after each level for better visibility.
2. What would happen if one of the level procedures failed?
    - Answer: If a level procedure fails (e.g., due to an error), the game may not proceed to the next level, and the player would not be able to complete the game.

---

3. Setting Procedure

Code Explanation:

```
setting PROC
    mWrite<" 1-change Color",0dh,0ah>
Again:
    mWrite<"Enter Choice:",0>
    mov eax,0
    call readdec
```

- What Happens: This procedure allows the user to change the text color by presenting a menu.
- Key Features: It uses a loop to prompt the user until a valid color choice is made.

Potential Questions:

1. Explain how the program changes the text color.
    - Answer: The program presents a list of color options to the user. Based on the user's input, it calls the `settextcolor` function with the corresponding color value.
2. What happens if the user inputs a number that is not associated with a color?
    - Answer: If the input is invalid, the program displays an error message and prompts the user to enter a valid choice again.

---

4. Level Procedures (level1, level2, level3)

Code Explanation (Example from `level1`):

```
level1 PROC
whileloop:
    cmp lives,0
    je quit
    mWrite<"Lives:",0>
    movzx eax,lives
    call writedec
    ...
    mov edx,OFFSET input
    mov ecx,9
    call ReadString
    ...
    mWrite<"You enter word not found!",0dh,0ah>
    dec lives
next:
    MOV EAX,500
    CALL delay
    call clrscr
    MOV AL,score
    cmp al,5
    jl whileloop
quit:
    ret
level1 endp
```

- What Happens: This procedure manages the gameplay for level 1, checking for lives and updating the score based on user input.
- Key Features: It includes loops for gameplay, checks for lives, and updates the score based on correct guesses.

Potential Questions:

1. ** How does the program determine if the player has lost a life?**

- Answer: The program checks the `lives` variable. If `lives` is equal to zero, it jumps to the `quit` label, indicating that the player has lost all lives.

2. Describe the process of checking if the entered word is correct.
    - Answer: The program reads the user's input and compares it against the correct answer. If the input does not match, it displays a message indicating the word was not found and decrements the `lives` variable.

---

5. File Handling Procedures

Code Explanation (Example from `read_file`):

```
read_file proc
    call OpenInputFile
    mov fileHandle,eax
    cmp eax,INVALID_HANDLE_VALUE
    jne file_ok
    mWrite <"Cannot open file",0dh,0ah>
    jmp quit
file_ok:
    mov edx,OFFSET buffer
    mov ecx,BUFFER_SIZE
    call ReadFromFile
    ...
    call CloseFile
quit:
    ret
read_file endp
```

- What Happens: This procedure attempts to open a file and read its contents into a buffer. It handles errors if the file cannot be opened.
- Key Features: It checks for errors during file operations and ensures proper resource management by closing the file after reading.

Potential Questions:

1. What are the steps involved in reading a file in this program?

- Answer: The program first calls `OpenInputFile` to open the file. It checks if the file handle is valid. If valid, it reads the file contents into a buffer using `ReadFromFile` and then closes the file with `CloseFile`.

2. How does the program handle errors related to file operations?
    - Answer: If the file cannot be opened (indicated by an invalid file handle), the program displays an error message and jumps to the `quit` label to exit the procedure gracefully.

---

6. Instruction Procedure

Code Explanation:

```
instruction PROC
    mov edx,offset file_L4
    call read_file
    call crlf
ret
instruction endp
```

- What Happens: This procedure reads and displays the instructions for the game from a file.
- Key Features: It utilizes the `read_file` procedure to fetch the instructions and then calls `crlf` to move to the next line after displaying them.

Potential Questions:

1. What is the purpose of the `instruction` procedure?
    - Answer: The `instruction` procedure is designed to read the game instructions from a file and display them to the user, ensuring they understand how to play the game.
2. How does the program ensure that the instructions are displayed correctly?
    - Answer: The program uses the `read_file` procedure to read the instructions into a buffer, and then it displays the contents of that buffer to the user, ensuring proper formatting with line breaks using `crlf`. ### 7. High Score Procedure

Code Explanation:

```
high_score PROC
    call clrscr
    mWrite<"High Scores:",0dh,0ah>
    call display_scores
    call crlf
    ret
high_score endp
```

- What Happens: This procedure clears the screen, displays the high scores, and calls another procedure to handle the display of scores.
- Key Features: It provides a way for users to view the top scores achieved in the game.

Potential Questions:

1. What is the purpose of the `high_score` procedure?
    - Answer: The `high_score` procedure is responsible for displaying the high scores of the game, allowing players to see their performance compared to others.
2. How does the program retrieve and display the high scores?
    - Answer: The program calls the `display_scores` procedure, which handles the logic for retrieving the high scores from storage and displaying them on the screen.

---

8. Quit Procedure

Code Explanation:

```
quit PROC
    mWrite<"Thank you for playing!",0dh,0ah>
    call delay
    call clrscr
    ret
quit endp
```

- What Happens: This procedure displays a thank-you message to the user and clears the screen before exiting the program.
- Key Features: It provides a graceful exit from the game, ensuring the user feels acknowledged.

Potential Questions:

1. What does the `quit` procedure do when the game ends?
    - Answer: The `quit` procedure displays a thank-you message to the player, waits for a brief moment, and then clears the screen before terminating the program.
2. Why is it important to provide a message before quitting the program?
    - Answer: Providing a message before quitting enhances user experience by acknowledging their participation and giving them a sense of closure after playing the game.

---

9. Error Handling

Code Explanation:

```
error_handling PROC

    mWrite<"An error has occurred!",0dh,0ah>

    call log_error

    ret

error_handling endp
```

- What Happens: This procedure displays an error message and calls another procedure to log the error for debugging purposes.
- Key Features: It ensures that any errors encountered during execution are communicated to the user and logged for further analysis.

Potential Questions:

1. How does the program handle errors?
    - Answer: The program uses the `error_handling` procedure to display an error message and log the error details, allowing developers to troubleshoot issues effectively.
2. What is the significance of logging errors?
    - Answer: Logging errors is crucial for debugging and maintaining the program, as it provides insights into what went wrong and helps developers fix issues in future versions.

---

10. Conclusion

```
conclusion PROC
    mWrite<"Game Over! Final Score: ",0>
    mov eax, final_score
    call writedec
    call crlf
    ret
conclusion endp
```

- What Happens: This procedure displays the final score of the game to the user, indicating the end of the game session.
- Key Features: It summarizes the player's performance and provides closure to the gaming experience.

Potential Questions:

1. What does the `conclusion` procedure display?
    - Answer: The `conclusion` procedure displays a message indicating that the game is over, followed by the player's final score.
2. Why is it important to show the final score?
    - Answer: Displaying the final score allows players to reflect on their performance, encourages competition, and can motivate them to improve in future sessions. ### 1. Initialization and Main Menu