Workshop Outline: C - Functions, Nested Loops

Title:

C - Functions and Nested Loops

Subtitle:

Session Outline

Welcome and Overview

- Brief introduction to the session objectives.
- Overview of the ALX project requirements and expected outcomes.

Objectives:

- Understand and implement functions in C.
- Utilize nested loops effectively.
- Differentiate between function declarations and definitions.
- Use header files and prototypes correctly.
- Apply coding standards and best practices.

Functions in C

Definition and Purpose:

- A function is a block of code that performs a specific task.
- Functions help in organizing code, reducing redundancy, and improving readability.

Function Declaration vs. Definition:

- **Declaration:** Specifies the function's name, return type, and parameters.

  ```c
  int add(int a, int b);
  ```

- **Definition:** Provides the actual body of the function.

  ```c
  int add(int a, int b) {
      return a + b;
  }
  ```

Function Prototypes:

- Prototypes declare functions before they are used, ensuring the compiler knows about the function's return type and parameters.

Scope of Variables:

- Local Variables: Declared within a function.
- Global Variables: Declared outside all functions, accessible to all functions in the file.

Slide 4: Basics of Functions and Nested Loops (20 minutes) Continued

---

Nested Loops

Definition and Use Cases:

- A nested loop is a loop inside another loop.
- Commonly used for multidimensional array processing, pattern printing, and complex iteration.

Syntax and Structure:

- Outer loop runs first, then the inner loop runs completely for each iteration of the outer loop.

  ```
  for (int i = 0; i < 5; i++) {
      for (int j = 0; j < 5; j++) {
          // Code to execute
      }
  }
  ```

Practical Examples of Nested Loops:

- Multiplication tables.
- Printing patterns like triangles, squares, etc.
- Iterating through matrices.

Task 1: Print Uppercase Alphabet

Objective:

Write a function that prints the alphabet in uppercase, followed by a new line.

Prototype:

```
void print_uppercase_alphabet(void);
```

Instructions:

- Use a loop to iterate through the uppercase alphabet.
- Utilize the ASCII values for uppercase letters.
- You can only use `_putchar` twice in your code.

Example:

```
A
B
C
...
Y
Z
```

Task 2: Print Numbers 0 to 9 Ten Times

Objective:

Write a function that prints numbers from 0 to 9, ten times, followed by a new line each time.

Prototype:

```
void print_numbers_x10(void);
```

Instructions:

- Use nested loops to achieve this task.
- Outer loop to control the number of times (10) and the inner loop to print numbers from 0 to 9.
- You can only use `_putchar` twice in your code.

Example:

```
0123456789
0123456789
...
0123456789
```

Task 3: Check for Uppercase Character

Objective:

Write a function that checks for an uppercase character.

Prototype:

```
int _isupper(int c);
```

Instructions:

- Use the ASCII values to determine if a character is uppercase.
- Returns 1 if `c` is uppercase, 0 otherwise.

Example:

```
_isupper('H') -> 1
_isupper('h') -> 0
```

Task 4: Print Multiplication Table up to n

Objective:

Write a function that prints the multiplication table up to `n`.

Prototype:

```
void print_multiplication_table(int n);
```

Instructions:

- If `n` is greater than 10 or less than 0 the function should not print anything.
- Use nested loops to generate the table.
- Ensure proper formatting for single and double-digit numbers.

Example:

```
print_multiplication_table(3) ->
0 1 2 3
0 1 2 3
0 2 4 6
0 3 6 9
```

Conclusion

Recap:

- Summary of the session highlights.
    - Understanding and implementing functions in C.
    - Utilizing nested loops effectively.
    - Differentiating between function declarations and definitions.
    - Using header files and prototypes correctly.
    - Applying coding standards and best practices.