COS284 Practical Assignment 4: Working with Structures in YASM Assembly Language

University of Pretoria
Department of Computer Science

Online Publication: 27 September 2024

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Plagiarism Policy

- All work submitted must be your own. You may not copy from classmates, textbooks, or other resources unless explicitly allowed.
- You may discuss the problem with classmates, but you may not write or debug code with other classmates, other than those in your group.
- If you use any external material, you must cite these sources.

Assignment Overview

In this assignment, you will demonstrate your practical knowledge of YASM assembly language, focusing on the use of structures (`structs`). You will complete three tasks that involve defining structs, manipulating data within them, and implementing a simple singly linked list. Each task builds upon the previous one, enhancing your understanding of how structs operate at the assembly level.

Assignment Objectives

The primary objectives of this assignment are to develop your understanding and skills in using structs in assembly language. Specifically, you will:

1. **Define and Use a Struct:**
    - Define a `Student` struct with specific fields.

- Initialize an instance of the struct.

2. **Work with an Array of Structs:**
   - Create an array of `Student` structs.
3. **Implement a Simple Linked List:**
   - Define a `StudentNode` struct with specific fields.
   - Manually link multiple `Student` instances to form a singly linked list.

Task 1: Creating and Displaying a Student Struct Using a Function

Objective

Create a function that allocates memory for a new `Student` struct using `malloc`, initializes the struct fields based on variables passed into the function call and return a pointer to the struct.

Instructions

1. **Define the `Student` Struct:**
   - Create a `Student` struct with the following fields:
     (a) `id`: 32-bit integer
     (b) `name`: Fixed-size character array of 64 bytes
     (c) `gpa`: 32-bit floating-point number
2. **Implement the Functionality:**
   - The function should:
     (a) Dynamically allocate memory for a new `Student` struct using `malloc`.
     (b) Accept the following parameters:
       – `id`: 32-bit integer
       – `name`: A string (fixed-size character array, passed as a pointer)
       – `gpa`: 32-bit floating-point number
     (c) Initialize the struct's fields with the provided parameters.
     (d) Return a pointer to the initialized `Student` struct.

Expected Outcome

An assembly program that defines a `Student` struct, implements the `createStudent` function, and returning a pointer to the allocated memory.

Implementation Details

- Write your assembly code in the file `task1_student_struct.asm`.
- Ensure your function correctly accepts and handles all parameters.

Example

If the function `createStudent` is called with the following parameters:

- `id`: 12345
- `name`: "John Doe"
- `gpa`: 3.75

The program should:

1. Dynamically allocate memory for a new `Student` struct.
2. Initialize the struct's fields with the provided data.
3. Return a pointer to the allocated `Student` struct.

Task 2: Adding a New Student Struct to an Array

Objective

Create a function that adds a new `Student` struct to an existing array of `Student` structs based on variables passed into the function call. The new `Student` will have an `id` that is one more than the highest `id` in the array. The function will receive the `name` and `gpa` as parameters, along with a pointer to the array and the array's maximum size.

Instructions

1. **Define the Function:**
   - Your function will receive the following parameters:
     (a) A pointer to the array of `Student` structs.
     (b) The maximum size of the array.
     (c) The name of the new student (string).
     (d) The GPA of the new student (float).
   - The function should add the new `Student` to the array if there is available space.
2. **Implement the Functionality:**
   - The function should:
     (a) Check if the array has space for a new `Student` (i.e., current number of students is less than the maximum size).

(b) If space is available:
   i. Iterate over the array to find the highest existing `id`.
   ii. Assign the new `Student`'s `id` to be one more than the highest `id` found.
   iii. Set the new `Student`'s `name` and `gpa` using the passed parameters.
   iv. Add the new `Student` to the array at the next available position.

(c) You may assume that we won't try and insert into an already full array

Expected Outcome

An assembly program that defines a function to add a new `Student` to an array based on provided parameters, ensures the `id` is correctly assigned, updates the array, and displays all `Student` structs, including the newly added one.

Implementation Details

- Write your assembly code in the file `task2_student_array.asm`.
- Ensure your function correctly accepts and handles all parameters.
- Use appropriate logic to determine the highest `id` in the array.
- Properly manage array indices and memory to avoid overwriting existing data.
- Include error handling for cases where the array has reached its maximum capacity.
- When displaying the students, ensure that only valid entries up to the current number of students are shown.

Example

Suppose the existing array can hold up to 3 students and currently contains:

```
Student 1:
ID: 12345
Name: John Doe
GPA: 3.75

Student 2:
ID: 67890
Name: Jane Smith
GPA: 3.85
```

If the function `add_student` is called with the following parameters:

- Name: "Alice Johnson"
- GPA: 3.95

The program should:

1. Check that the array has space (current number of students $<$ maximum size).

2. Find the highest `id` in the array (67890).
3. Assign the new `id` as 67891.
4. Add the new `Student` to the array.

Task 3: Implementing a Linked List Using `StudentNode` Struct

Objective

Create a singly linked list using a separate `StudentNode` struct that contains a `Student` object and a `next` pointer. Implement a function that adds a new `Student` to the linked list based on variables passed into the function call. The new `Student` will have an `id` that is one more than the highest `id` in the linked list. The function will receive the head of the linked list, as well as the `name` and `gpa` of the new student.

Instructions

1. **Define the `StudentNode` Struct:**
   - Create a new struct called `StudentNode` that contains:
     (a) A `Student` object.
     (b) A `next` field to hold a pointer to the next `StudentNode`.
   - `next`: 64-bit pointer.
2. **Define the Function:**
   - Your function will be passed the following parameters:
     (a) A pointer to the head of the linked list (`StudentNode` struct).
     (b) The name of the new student (string).
     (c) The GPA of the new student (float).
   - The function should add the new `StudentNode` to the linked list.
3. **Implement the Functionality:**
   - The function should:
     (a) Traverse the linked list to find:
       i. The largest existing `id` in the `Student` objects.
       ii. The last `StudentNode` in the list (where `next` is null).
     (b) Assign the new `Student`'s `id` to be one more than the highest `id` found.
     (c) Create a new `StudentNode` with:
       i. The `Student` object containing the new `id`, `name`, and `gpa`.
       ii. `next` pointer set to null.
     (d) Update the `next` pointer of the last `StudentNode` to point to the new `StudentNode`.

Expected Outcome

An assembly program that defines a `StudentNode` struct to create a linked list of students. The program includes a function to add a new `Student` to the linked list based on provided parameters, ensures the `id` is correctly assigned, updates the linked list, and displays all `Student` structs, including the newly added one.

Implementation Details

- Write your assembly code in the file `task3_student_linkedlist.asm`.
- Ensure your function correctly accepts and handles all parameters.
- Use appropriate logic to determine the highest `id` in the linked list.
- Properly manage memory allocation for the new `StudentNode`. Since dynamic memory allocation is available, ensure the correct amount of memory with padding for the structs are allocated.
- Pay careful attention to pointer sizes and memory addresses.

Example

Suppose the existing linked list contains:

```
Student 1:
ID: 12345
Name: John Doe
GPA: 3.75

Student 2:
ID: 67890
Name: Jane Smith
GPA: 3.85
```

If the function is called with the following parameters:

- Head pointer: pointer to the first `StudentNode` (containing John Doe).
- Name: "Alice Johnson"
- GPA: 3.95

The program should:

1. Traverse the linked list to find:
   (a) The largest `id` in the `Student` objects (67890).
   (b) The last `StudentNode` (which contains Jane Smith).
2. Assign the new `id` as 67891.

3. Create a new `StudentNode` with:
    - `Student` object:
        - `id`: 67891
        - `name`: "Alice Johnson"
        - `gpa`: 3.95
    - `next`: null
4. Update the `next` pointer of the last `StudentNode` (Jane Smith's node) to point to the new `StudentNode`.

Struct Definitions

For clarity, the structs can be defined as:

- `Student` **Struct**
    - `id`: 32-bit integer
    - `name`: Fixed-size character array of 64 bytes
    - `gpa`: 32-bit floating-point number
- `StudentNode` **Struct**
    - `student`: 64-bit pointer to `Student` struct
    - `next`: 64-bit pointer to `StudentNode`

Mark Distribution

The total marks for this assignment are 20, with the distribution as follows:

- **Task 1: Defining and Using a Struct** (6 marks)
    - Correct definition of the `Student` struct: **2 marks**
    - Proper initialization and display of the struct: **3 marks**
- **Task 2: Array of Structs** (7 marks)
    - Correct creation and initialization of the array: **2 marks**
    - Proper iteration and display of each `Student`: **3 marks**
- **Task 3: Implementing a Simple Linked List** (7 marks)
    - Correct modification of the `Student` struct to include the `next` pointer: **2 marks**
    - Proper linking, traversal, and display of the linked list: **3 marks**

Submission

- Submit your assembly source code files for each task:
  - `task1_student_struct.asm`
  - `task2_student_array.asm`
  - `task3_student_linkedlist.asm`

Assessment Criteria

- **Correctness:** The program functions as specified in each task.
- **Use of Structs:** Proper definition and manipulation of structs.

Additional Resources

- YASM Documentation: https://yasm.tortall.net/
- YASM tutorial playlist: Playlist

Academic Integrity

Remember to adhere to the university's policies on academic integrity. Any form of plagiarism or academic dishonesty will be dealt with according to university regulations.