# Assignment 3: Stack and Heap Memory Management

Razi Haider

19 November 2023

## 1 Introduction

The purpose of this report is to provide a comprehensive explanation of the C code created for the Operating Systems project on Stack and Heap Memory Management. The goal of the assignment is to simulate a memory system with a basic stack and heap. The allocated memory size is 500 bytes, which is divided into specific stack and heap regions.

## 2 Memory Layout

### 2.1 Stack Memory Layout

The stack memory starts with a capacity of 100 bytes and can expand up to a maximum of 300 bytes. It consists of stack frames and frame statuses, where each frame status is 21 bytes in size and holds important information about the stack frames. This information includes the frame number, function name, function address, frame address, and a boolean value indicating whether it's the top of the stack.

Following the frame status list in the stack, there are up to 5 stack frames, each with a range of data types including integers, doubles, characters, and data pointers. A stack frame can have a size between 10 to 80 bytes.

### 2.2 Heap Memory Layout

The maximum size of heap memory is 300 bytes. When a region is allocated on the heap, an additional 8 bytes are used to store its size and a randomly generated magic number. The system keeps track of available memory segments by maintaining a free list.

## 3 Code Structure

### 3.1 Structures

- **frame_status_t:** This structure is used to store the status of the frame, whether it is used or not, the function address, frame address and the name of the function.
- **freelist_t:** This structure is used to store the free list, it stores the start address of the free block, the size of the free block and a pointer to the next free block.
- **allocated_status_t:** This structure is used to store the allocated buffer, it stores the name of the buffer, the start address of the buffer and the size of the buffer
- **int_t:** This structure is used to store the integer, it stores the name of the integer, the value of the integer and a flag to check if the integer has been initialized or not.
- **double_t:** This structure is used to store the double, it stores the name of the double, the value of the double and a flag to check if the double has been initialized or not.

- **char_t:** This structure is used to store the char, it stores the name of the char, the value of the char and a flag to check if the char has been initialized or not.
- **frame_t:** This structure is used to store the frame, it stores the frame address, the size of the frame, and the integer, double, char and pointer arrays.
- **memory_t:** This structure is used to store the memory, it stores the frame status, stack frame, free list, stack size, heap size and the heap.

### 3.2 Functions

- **init:** This function initializes the memory system, setting up frame statuses, stack frames, the free list, and heap-related variables. It ensures a clean slate for subsequent operations.
- **CF:** This function is used to create a new frame, it checks if the function name is valid, if the stack size is less than the maximum stack size and if the function already exists.
- **DF:** This function is used to delete a frame, it checks if the stack is empty, if the frame exists and then deletes the frame.
- **CI, CD, CC:** These functions are used to Check for the existence of frames and available space within the current frame and Create variables of the respective types in the current frame.
- **CH:** This function allocates memory on the heap for character buffers, updating heap-related information and enabling dynamic memory management.
- **SM:** This function displays the current state of the stack and heap memory, offering a detailed snapshot for debugging and understanding the memory structure.
- **main:** This function manages user interaction, providing a command-line interface for executing memory management operations based on user input.

## 4 Conclusion

The code implements a simple memory management system with stack frames and heap memory. It follows a procedural approach, with functions responsible for specific tasks such as frame creation, deletion, variable creation, heap allocation, and memory state printing. The data structures provide a structured representation of the memory system, facilitating operations and information retrieval. The main function of the system is to manage user interactions and process commands., creating an interactive environment for users to manipulate the memory system.