Ministry of Education of Republic of Moldova

Technical University of Moldova

Faculty of Computers, Informatics and Microelectronics

Software Engineering Department

Computer Programming

Laboratory work #5

Pointers and dynamically memory allocation

Author:
Alexandru Rudoi
std. gr. FAF-231

Verified:
Alexandru Furdui

Chișinău 2023

Theory Background

1. **Data Structure (Stack):** A stack is a data structure that follows the Last-In, First-Out (LIFO) principle. It means the last element added is the first to be removed.

2. **Menu-Driven Interface:** This refers to a user interface that allows users to interact with a program by selecting options from a menu displayed on the screen.

3. **Data Types Enumeration (DataType):** An enumeration is a list of named values. In this case, DataType enumerates the types of data that can be stored in the stack, such as integers, characters, floats, and user-defined structures.

4. **User-Defined Struct (UserStruct):** A user-defined structure is a custom data structure created by the programmer. In this code, it's used to define a structure with two fields: an integer 'id' and a character array 'name'.

5. **ANSI Escape Codes:**

   These are special character sequences used in text to control text formatting and color in a console or terminal. In this context, they are used to display colored text.

6. **Functions:** Functions are blocks of code that perform specific tasks. In this code, functions are used for various operations, such as creating new stack nodes, pushing data onto the stack, popping data from the stack, and more.

7. **Clear Screen Function:** This function clears the contents of the console screen, providing a clean slate for displaying information.

8. **User Input Parsing:**

   This refers to the process of interpreting and extracting meaningful information from user-provided input.

9. **Dynamic Memory Allocation (malloc and free):**

   - *malloc* is a function used to allocate memory dynamically during program execution. It's used to create memory for stack nodes.

   - *free* is used to release memory that was previously allocated with *malloc*.

The Task

Implement your own stack data structure in C using dynamic memory allocation and pointers. A stack is a linear data structure that follows the Last-In, First-Out (LIFO) principle. Your implementation should provide the following functionalities:

• Push: Add an element to the top of the stack.

• Pop: Remove and return the element from the top of the stack.

• Peek: Return the element at the top of the stack without removing it.

• IsEmpty: Check if the stack is empty.

• IsFull: Check if the stack is full (if there is a maximum capacity, but there should be one so choose one properly).

Task Requirements:

• Implement a stack data structure using dynamic memory allocation (malloc and free) and pointers to store the elements.

• Ensure that the stack can hold elements of any data type (integers, characters, floats or optionally user-defined structs).

• Define appropriate data structures (e.g. linked lists or arrays) to store the elements in the stack.

• Properly manage memory allocation and deallocation when pushing and popping elements from the stack.

• Ensure proper error handling, such as checking for stack underflow (trying to pop from an empty stack).

• Provide a demonstration in the main program that showcases the stack's functionality. For example, you can push and pop elements to/from the stack and display its contents.

• Test your stack implementation with various scenarios to ensure it functions correctly and handles different cases gracefully.

Technical implementation

Listing of the program: Github

Pseudo-code:

```
Define DataType Enumeration:
    - Fields: MY_INT, MY_CHAR, MY_FLOAT, MY_USER_STRUCT

Define UserStruct:
    - Fields: id (integer), name (character array)

Define DataWithDataType Structure:
    - Fields: type (DataType), data (union of int, char,
       float, UserStruct)
```

```
Define StackNode Structure:
    - Fields: data (DataWithDataType), next (pointer to
      StackNode)

Function newNode(data):
    Create a new StackNode
    Set data field with the provided data
    Set next to NULL
    Return the new StackNode

Function push(top, data, maxStackSize):
    If the stack is full (isFull(top, maxStackSize, size_of(
      DataWithDataType)):
        Print an error message
        Return newNode

Function pop(top, poppedData):
    If the stack is empty (isEmpty(top)):
        Return
    Get the top node and store its data in poppedData
    Update the top to the next node
    Free the memory of the popped node

Function peek(top):
    If the stack is empty (isEmpty(top)):
        Return
    Print the data of the top node

Function isEmpty(top):
    Return true if top is NULL, else false

Function isFull(top, maxStackSize, dataSize):
    Count the current size of the stack
    If the total size is greater than or equal to (
      maxStackSize * dataSize), return true; otherwise,
      return false
```

Results

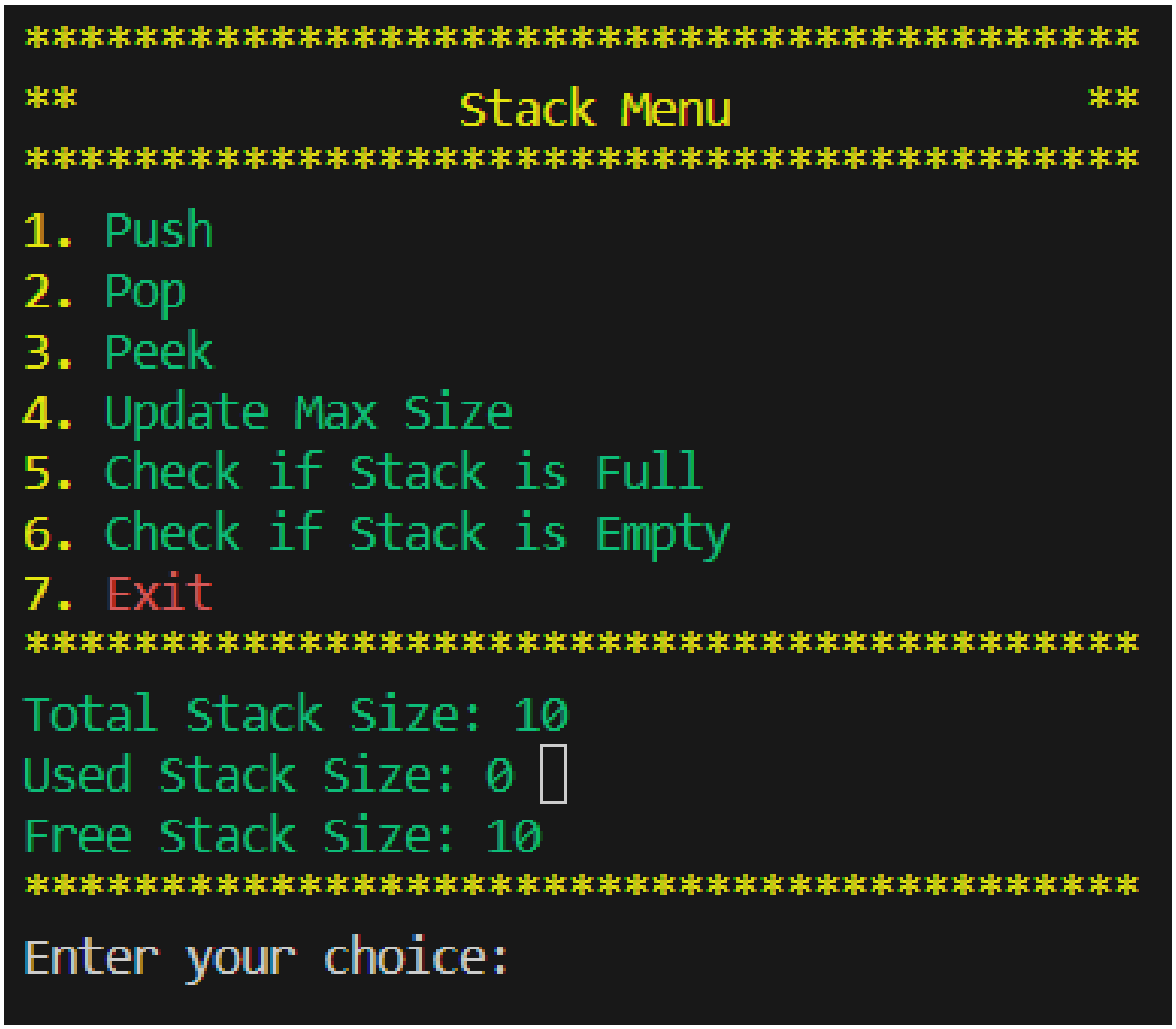

Figure 1

Conclusion

In conclusion, this program has been a valuable learning experience that has enriched my understanding of data structures, user interface design, and best practices in programming. It has equipped me with the foundational knowledge and skills that will undoubtedly benefit me both as a student and as a future programmer in various software development endeavours.

Bibliography

1. The overview of Computer Programming course lessons for students (lecturer: associate professor M. Kulev). Chisinau, UTM, FCIM, 2023.
2. https://www.digitalocean.com/community/tutorials/stack-in-c
3. https://www.geeksforgeeks.org/introduction-to-stack
4. https://www.programiz.com/dsa/stack
5. https://www.javatpoint.com/data-structure-stack