COS284 Assignment 3

University of Pretoria
Department of Computer Science

Online Publication: 4 September 2023

Plagiarism Policy

- All work submitted must be your own. You may not copy from classmates, text-books, or other resources unless explicitly allowed.
- You may discuss the problem with classmates, but you may not write or debug code together.
- If you use any advanced material, you must cite these sources.

Task

In this task you implement a simple library filled with books. The books and library will be defined by structs, and allocated dynamically.

Books

You need to implement a book struct, which contains the following fields:

```
struct Book
{
  char isbn[13];
  char title[50];
  float price;
  int quantity;
};
```

You will need to write an assembly function that receives the necessary parameters to create a book struct, and returns a pointer to the newly created and populated struct. The function signature is as follows:

```
struct Book* allocateBook(char* isbn, char* title, float
   ↪ price, int quantity);
```

The struct and the function must be implemented in a single file called `book.asm`.

Library

You need to implement a library struct, which contains the following fields:

```
struct Library
{
  struct Book books[5];
  int count;
};
```

You will need to write a function that initialises an empty library with 0 books.

```
struct Library* initialiseLibrary(void);
```

The pointer returned from this function should have count set to 0, and the books array should be empty. To add a book to the library, you need to implement the following function:

```
int addBook(struct Library *lib, struct Book *book);
```

This function should add the book to the library, and return 1 if the book was added successfully, and 0 if the book could not be added. The library should not be able to contain more than 5 books. The function should also not add a book if the ISBN already exists in the library, it should instead add the quantity of the book with the same ISBN. The library can only hold 5 unique books, however it can hold multiple copies of the same book.

- A deep copy must be made of the book

To search for a book in the library, you need to implement the following function:

```
struct Book* searchBookByISBN(struct Library *lib, char *isbn
   ↪ );
```

This function should return a pointer to the book if it is found, and a null pointer if the book is not found.

Additional Notes

For this assignment you may use any C standard library functions to assist you. The recommended options to consider are:

- `malloc`
- `strcpy`

- `strcmp`

With that in mind, you may assume all strings will be cstrings - they will be null terminated:

```
char isbn[13] = "978-3-16-148\0";
```

You should be using stack memory for storing your local variables and initial parameters. Using the Visual Studio Code Watch pane, you can view the ISBN on the stack with:

```
*(char*)*(void**)$rsp@13
```

This is assuming your stack pointer currently points to the memory location of the ISBN string parameter. The additional (void**) coercion is required to retrieve the address of the string, since $rsp will store the address that holds the address of the string. The @13 is required to indicate that the string is 13 bytes long.

Correctly using the stack

Take note that any additional pushing or popping will change the stack pointer, and make it difficult to keep track of your local variable offsets. Both Figure 1 and Figure 2 are

```
function:
push rbp
mov rbp, rsp

push rdi
push rsi

; do stuff

pop rsi
pop rdi

leave
ret
```

Figure 1: Push-Pop Mechanism

```
function:
.param1 equ 0
.param2 equ 8
push rbp
mov rbp, rsp

sub rsp, 16
mov [rsp + .param1], rdi
mov [rsp + .param2], rsi

; do stuff

leave
ret
```

Figure 2: Manual Stack Manipulation

valid forms of stack usage, however your approach for accessing local variables will differ. In Figure 1, you will effectively need to track the exact order and alignment of your push and pops. While in Figure 2, you will always know exactly where in the stack your local variables are located. The choice is yours, but you should be consistent in your approach.

Mixing the two approaches

If you have been using the two approaches of pushing and popping, while also manually allocating memory on the stack, you may have noticed that the offsets of your local variables are incorrect. This approach is very difficult to debug and has a lot of room for error. However, in the event that you do want to use it, I recommend accessing your local variable relative to the base pointer. The base pointer will hold the top of your current stack frame and will not change with additional push and pop instructions. **Take note that you will be subtracting from the base pointer, not adding to it**.

```
function:
.param1 equ 0
.param2 equ 8
push rbp
mov rbp, rsp

sub rsp, 16
mov [rbp - .param1], rdi
mov [rbp - .param2], rsi
push r15
; do stuff

pop r15
leave
ret
```

Mark Distribution

Task	**Marks**
Allocating Books	15
Initialising an empty library	5
Adding books to the library	30
Searching the library	10
Total	**60**