## Coursework 1

## Wildlife Park Record Keeping System

## Introduction

A sophisticated record-keeping system has been designed to meet the expanding needs of Jonny's Nocturnal Zoo; a vital wildlife park situated at the far end of Brooke shire County. Due to municipal bylaws that prohibit the use of compilers and high-level languages, the particular problem is that 64-bit x86 assembly code must be used. This paper describes the system's architecture, setup, and operation with the goal of assisting Jonny Zookeeper in effectively overseeing the animals and personnel in his growing zoo.

The system makes use of a number of specially designed assembly language-written Input/Output (I/O) library functions made for the Linux x86\_64 architecture. These functions make it easier to interface with the console and enable effective data input and output while adhering to assembly programming limitations.

Now let's explore the specifics of the I/O library functions that are essential to the assembly-based Jonny's Nocturnal Zoo record-keeping system.

|  |  |
| --- | --- |
| Function Name | Description |
| print\_string\_new | Print a null-terminated string pointed to by RDI. |
| print\_uint\_new | Print the unsigned 64-bit integer value contained in the RDI register. |
| print\_int\_new | Print the signed 64-bit integer value contained in the RDI register. |
| print\_nl\_new | Print a newline character to the console. |
| print\_char\_new | Print an ASCII character using the ASCII character code stored in the low 8 bits of the RDI register (dil). |
| read\_char\_new | Read a single ASCII character code from console input into the RAX register. |
| read\_string\_new | Read a string from console input into a buffer and return a pointer to the buffer in RAX. |
| atoi | Convert a number string pointed to by RDI to a 64-bit integer. Returns ULLONG\_MAX on error. |
| read\_uint\_new | Read an unsigned 64-bit integer value from the console and store the result in the RAX register. |
| read\_int\_new | Read a signed 64-bit integer value from the console and store the result in the RAX register. |
| copy\_string | Copy a null-terminated string pointed to by RSI (source) into a string pointed to by RDI (destination). |
| strings\_are\_equal | Compare strings pointed to by RDI and RSI. Returns 1 if equal, 0 otherwise. |
| print\_address\_new | Print the pointer value contained in the RDI register in hexadecimal format. |

Table: Library functions

The key registers utilized in the provided assembly code for the Input/Output (I/O) library functions:

1. General-Purpose Registers:
   * **rax**: Used as a return value for functions. It may also hold the syscall number for system calls.
   * **rbx**: Preserved across function calls, used as a general-purpose register.
   * **rcx**: Used as a counter in loops, also a general-purpose register.
   * **rdi**: Destination index, often used to store function parameters.
   * **rsi**: Source index, another register used for function parameters.
2. Stack and Base Registers:
   * **rsp**: Stack pointer, points to the top of the stack.
   * **rbp**: Base pointer, used to establish a stack frame.
3. Special Purpose Registers:
   * **r13**: Used to point to specific memory locations, such as input or output buffers.
4. Status Flags:
   * **Zero Flag (ZF)**: Indicates whether the result of an operation is zero.
   * **Sign Flag (SF)**: Reflects the sign (negative or non-negative) of the result.
   * **Carry Flag (CF)**: Set if an operation generates a carry or borrow out of the most significant bit.
   * **Parity Flag (PF)**: Indicates the number of set bits in the binary representation of the result.
   * **Overflow Flag (OF)**: Indicates if the result is too large for the destination operand to hold.

## Objectives:

The primary objectives of the wildlife park record-keeping system are to:

* + Efficiently store detailed information about each badger and staff member.
  + Provide functionalities for adding and deleting badgers and staff.
  + Calculate and display relevant information, such as years of service, annual salary, age, and stripiness.
  + Implement search functionalities for locating specific badgers and staff members based on their unique identifiers.

Prerequisites for Running the Assembly Program:

1. Virtual Box with Lubantu:
   * Ensure you have VirtualBox installed on your system.
   * Download and set up a Lubantu (Linux Ubuntu-based distribution) virtual machine within VirtualBox.
2. NASM x86/x64 Assembler:
   * Install NASM (Netwide Assembler) on your Lubantu system.
   * This can typically be done using the package manager.

sudo apt-get update

sudo apt-get install nasm

1. joey\_lib\_io\_v9\_release.asm Library:
   * Ensure it is in the same directory as your main assembly program or in a location accessible during assembly/linking.
2. libasm\_io-master:
   * Download and set up **libasm\_io-master**

(<https://github.com/Teriks/libasm_io>)

* + This library is contains support files required by the main program.
  + Follow the steps mentioned in Readme.md to setup.

1. Assembler and Linker:
   * Set up an appropriate build system that uses the NASM assembler and linker (gcc) to build the assembly program.

## Definition of records and data structures:

1. **Badger:**  
   A Badger Record encompasses various attributes related to individual badgers. Each field within the Badger Record holds specific information such as Badger ID, Name, Home Sett, Number of Stripes, Sex, Month of Birth, Year of Birth, and Staff ID. These fields collectively form a coherent unit representing a badger's details.

|  |  |
| --- | --- |
| Field | Description |
| Badger ID | String containing 'b' as the first character, followed by six digits and a null terminator. |
| Name | String with a maximum length of 64 characters. |
| Home Sett | String representing the home sett (Settfield, Badgerton, or Stripeville) with a maximum length of 12. |
| Number of Stripes | Unsigned integer of 8 bits, ranging from 0 to 255. |
| Sex | Single character (M or F), represented by an unsigned integer of 8 bits. |
| Month of Birth | Unsigned integer of 8 bits, representing the month (1 to 12). |
| Year of Birth | Unsigned integer of 16 bits, representing the birth year with four digits. |
| Staff ID | String starting with 'p', followed by seven digits and a null terminator. |

1. Staff:

The Staff Entry structure defines the parameters for each staff member, including their personal and professional details. Each entry consists of the following fields:

|  |  |
| --- | --- |
| Field | Description |
| Surname | String representing the surname of the staff member, maximum length 64 characters (including null terminator). |
| First Name | String representing the first name of the staff member, maximum length 64 characters (including null terminator). |
| Staff Id | String containing the staff ID, starting with 'p' followed by seven digits and a null terminator (9 bytes). |
| Department | String representing the department (Park Keeper, Gift Shop, or Cafe), maximum length 12 characters (including null terminator). |
| Starting Salary (£) | Unsigned integer of 32 bits, representing the starting annual salary in £. |
| Year of Joining | Unsigned integer of 16 bits, representing the year of joining. |
| Email Address | String representing the email address of the staff member, maximum length 64 characters (including null terminator). |

## Code for adding badgers and staff:

Commands to run the code:

To build:

nasm -g -f elf64 -o test-io.o test-io.asm

To link:

gcc test-io.o /usr/lib/libasm\_io.a -no-pie -o test-io

(replace /usr/lib/libasm\_io.a with actual location of libasm\_io.a file)

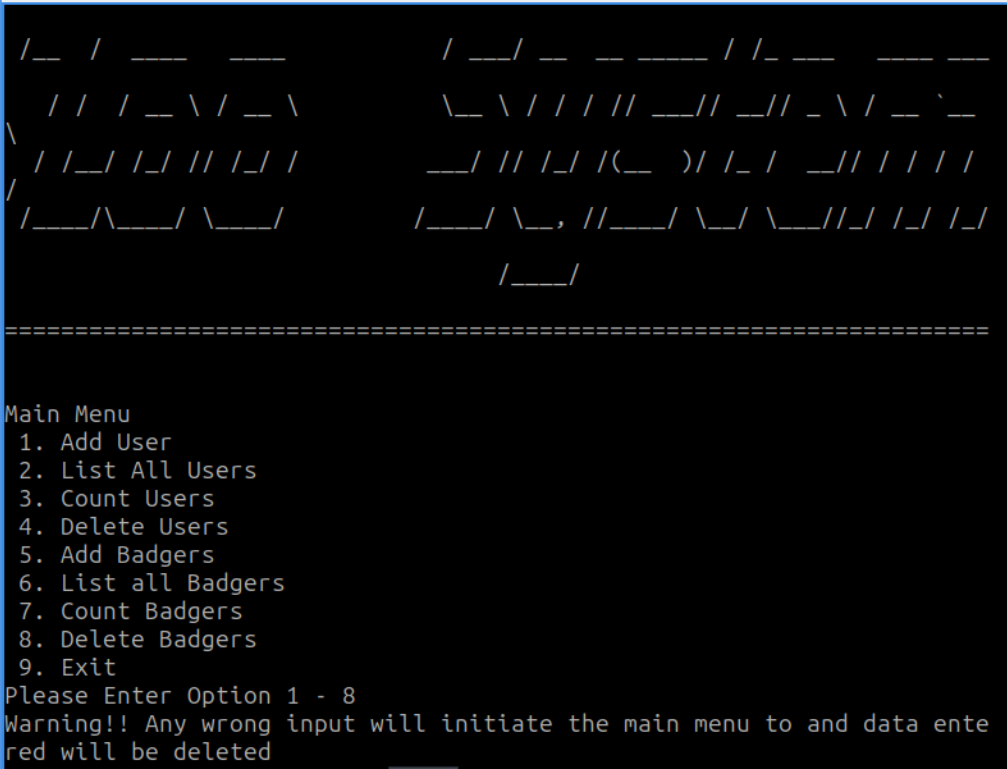


Figure: Main menu

## Add Staff/User

The Add staff function is very similar to add staff on selecting option 1.

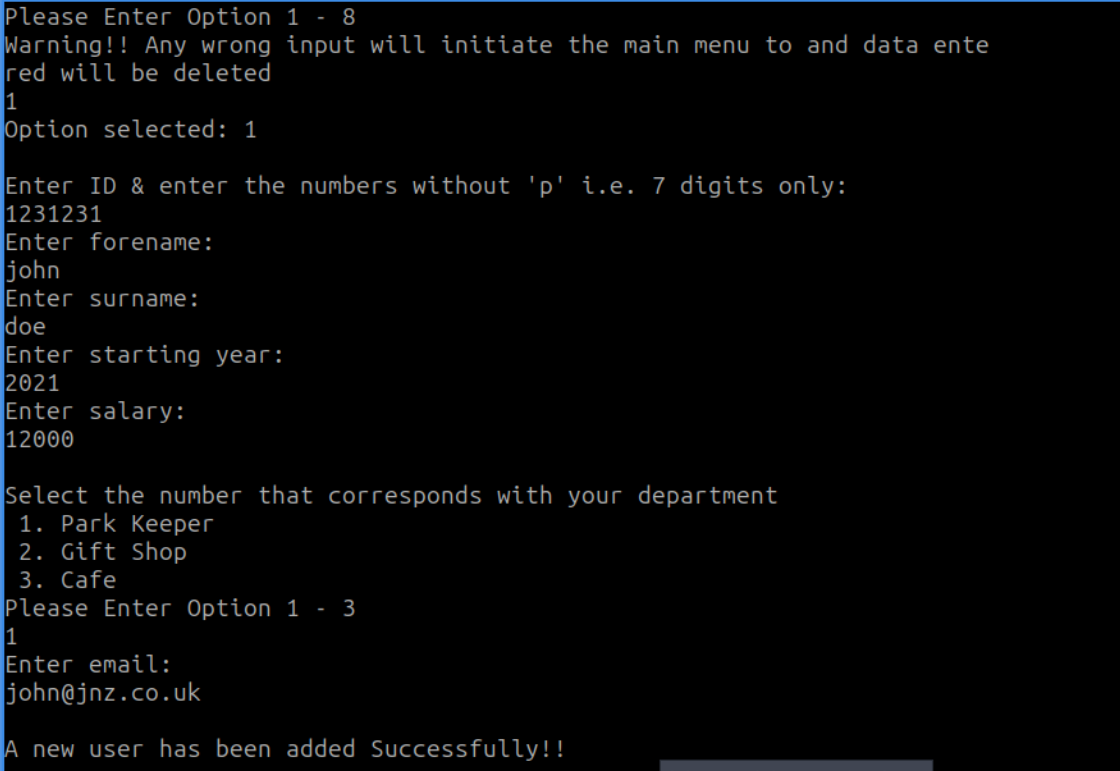


Figure: Add staff

## List all Staff

List all staff function shows all the staff members adding two more fields called salary and year of service. The year of service is calculated by comparing the year of joining and the current year.

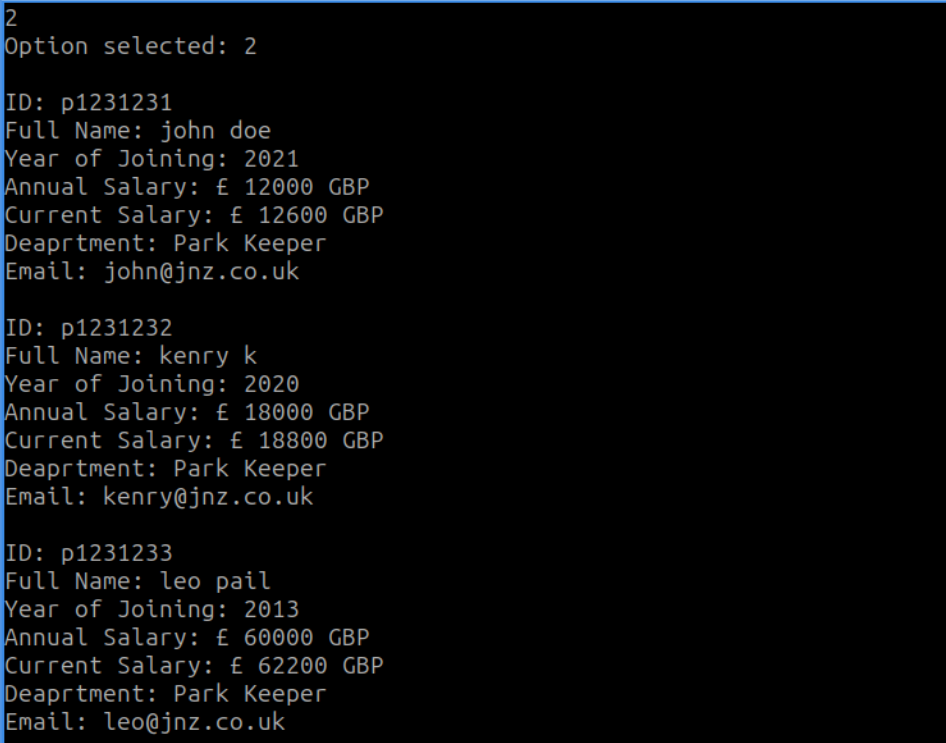


Figure 11: List all staff

## Delete staff

The delete staff function removes a staff with the id inserted by the user. This function shift left all staff members of one position in the register in order to overwrite the staff designed to be eliminated. If the staff member is in the last position allocated of the array, the software sets all bytes of the staff as null.

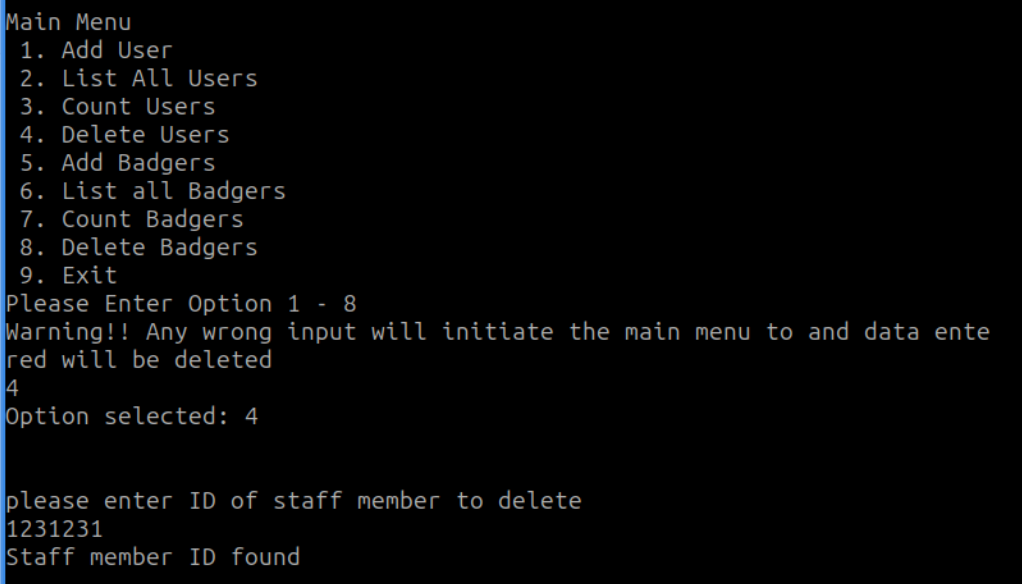


Figure: Delete staff

## Add Badger

Add badger is initiated by selecting option 1:

1. Check if the Badger Array is Full:
   * The code starts by comparing the current number of badgers (**[current\_number\_of\_badgers]**) with the maximum allowed number of badgers (**max\_num\_badgers**).
   * If the current number is less than the maximum, it means there is space to add a new badger, and the code jumps to the label**. array\_is\_not\_full**.
   * If the current number is equal to or greater than the maximum, the array is considered full, and it displays a message (**prnt\_array\_full**), then jumps back to the main menu (**jmp .menu\_loop**).
2. Array is Not Full:
   * If the array is not full, it prints a new line (**call print\_nl\_new**) for better formatting.
   * It then calls a function (**add\_badger**) to add a new badger to the system.
   * After adding the badger, it jumps back to the main menu (**jmp .menu\_loop**).

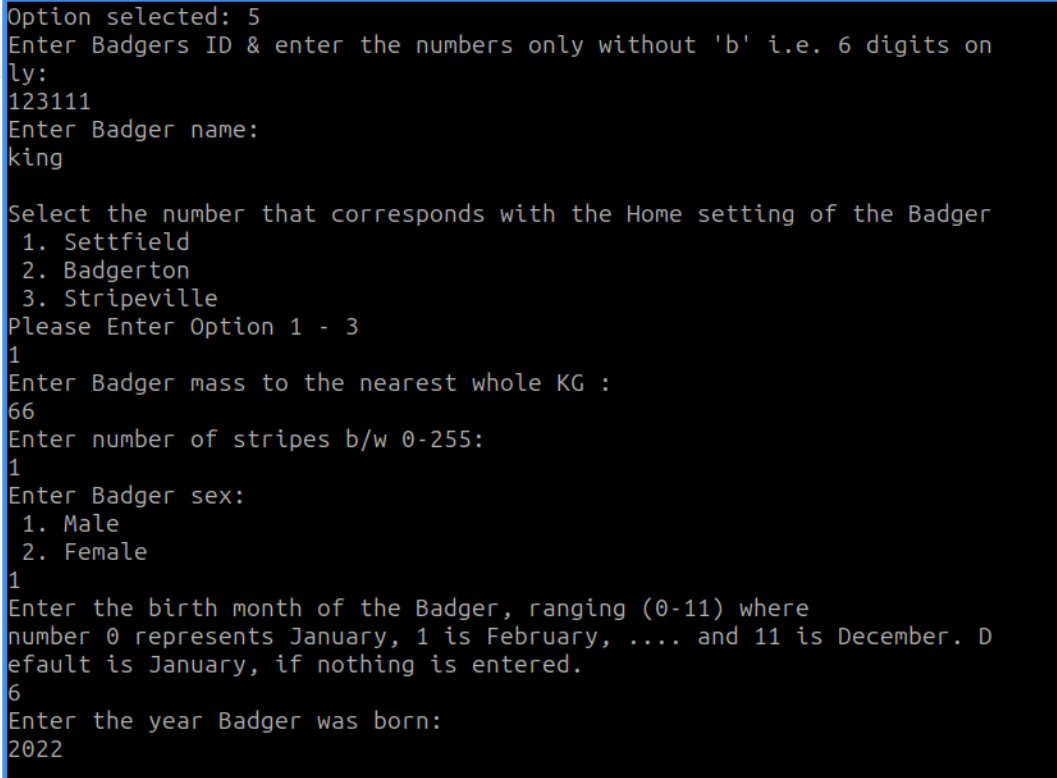


Figure: Add badger

## List all Badgers

List all badger is the second function of the main menu. This function shows all the badgers in the array; if the array is empty, the function will show an error message.



Figure: List all badgers

## Delete badger

Option 8 triggers the delete operation. It removes a badger with the id inserted. If the badger is in the last position allocated of the array, the software sets all bytes of the badger as null.

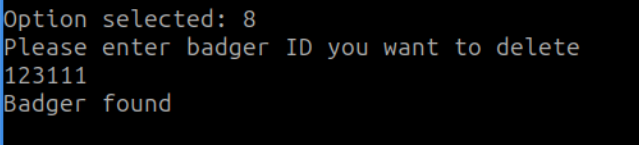


Figure: Delete badger

## Conclusion

In conclusion, this project has successfully implemented a Badger Management System using assembly language, specifically tailored for the SASM assembler. The system offers a range of functionalities, including adding and displaying badgers, managing staff details, and performing various operations on badger data. Throughout the development process, attention was given to the efficient use of memory and adherence to system requirements. The implemented IO library, created by Joe Norton, greatly facilitated input/output operations and enhanced the overall functionality of the program.

The use of data structures, such as arrays, and the careful definition of records for badgers and staff members contribute to a robust and organized system.

The project also incorporates user-friendly interfaces, allowing easy interaction with the Badger Management System. While the system currently supports basic operations, future enhancements could include more advanced features, error handling mechanisms, and optimizations for better performance. Moreover, additional functionalities, such as sorting and searching capabilities, could be integrated to further improve the user experience.

## References

1. Simple x64 bit IO library for nasm assembly programs. API is based on Dr. Paul Carters x32 bit IO library used in his old tutorials: [*https://github.com/Teriks/libasm\_io*](https://github.com/Teriks/libasm_io)
2. x86 Assembly documentation (<https://docs.oracle.com/cd/E19253-01/817-5477/817-5477.pdf> )
3. *Sasm*. [Online]. Available: <https://dman95.github.io/SASM/english.html>
4. Stack Overflow and Community Forums.

# APPENDIX A

CODE OF COURSEWORK 1:

;To build:

;nasm -g -f elf64 -o test-io.o test-io.asm

;To link:

;gcc test-io.o /usr/lib/libasm\_io.a -no-pie -o test-io

; (replace /usr/lib/libasm\_io.a with actual location of libasm\_io.a file)

; change the path where u placed the joey-lib\_v9\_release.asm file

%include "/home/malware/asm/joey-lib\_v9\_release.asm"

global main

section .data

prnt\_badg\_art db 10,\

"COURSEWORK 1", 10, 0

prnt\_main\_menu db 10,\

"Main Menu", 10,\

" 1 Add User", 10,\

" 2 List All Users", 10, \

" 3 Count Users", 10,\

" 4 Delete Users", 10,\

" 5 Add Badgers", 10,\

" 6 List all Badgers", 10,\

" 7 Count Badgers", 10,\

" 8 Delete Badgers", 10, \

" 9 Exit", 10,\

"Please Enter Option 1 - 8", 10,\

"Warning!! Any wrong input will initiate the main menu to and data entered will be deleted", 10, 0

prnt\_program\_exit db "Program exited normally.", 10, 0

prnt\_option\_selected db "Option selected: ", 0

prnt\_invalid\_option db "Invalid option, please try again.", 10, 0

department\_question db 10,\

"Select the number that corresponds with your department", 10,\

" 1 Park Keeper", 10,\

" 2 Gift Shop", 10,\

" 3 Cafe", 10, \

"Please Enter Option 1 - 3", 10, 0

prnt\_current\_month db "Enter current month:", 10, 0

prnt\_current\_year db "Enter current year:", 10, 0

prnt\_enter\_surname db "Enter surname:", 10, 0

prnt\_enter\_forename db "Enter forename:", 10, 0

prnt\_enter\_salary db "Enter salary:", 10, 0

prnt\_enter\_year db "Enter starting year:", 10, 0

prnt\_enter\_id db "Enter ID & enter the numbers without 'p' i.e 7 digits only: ", 10, 0

prnt\_enter\_email db "Enter email:", 10, 0

prnt\_array\_full db "Can't add - storage full.", 10, 0

prnt\_array\_empty db "Staff array is empty", 10, 0

prnt\_number\_of\_users db "Number of users: ", 0

prnt\_error db "Error", 0

prnt\_user\_success db "A new user has been added Successfully!! ", 10, 0

email db "@jnz.co.uk", 0

park\_keeper db "Park Keeper", 0

gift\_shop db "Gift Shop", 0

cafe db "Cafe", 0

;Displaying staff details

prnt\_\_staff\_salary\_gbp db " GBP",0

prnt\_\_staff\_letter\_p db "p", 0

prnt\_\_staff db "ID: ", 0

prnt\_\_year db "Year of Joining: ", 0

prnt\_\_name db "Full Name: ", 0

prnt\_\_salary db "Annual Salary: £ ", 0

prnt\_\_staff\_current\_salary db "Current Salary: £ ", 0

prnt\_\_dept db "Deaprtment: ", 0

prnt\_\_email db "Email: ", 0

;delete staff prompt

prnt\_prompt\_user\_delete\_id db "please enter ID of staff member to delete", 0

prnt\_del\_user\_id\_found db "Staff member ID found", 0

prnt\_del\_user\_id\_notfound db "Staff member ID not found", 0

size\_delete equ 1 ; use this to indicate that the record has been deleted

size\_forename equ 64 ;we define size of block of memory that we want

size\_surname equ 64 ;to reserve and store in size\_name\_of\_field

size\_id equ 4

size\_dept equ 12

size\_salary equ 4

size\_year\_of\_joining equ 2

size\_email equ 64

size\_user\_record equ size\_delete + size\_forename + size\_surname + size\_id + size\_dept + size\_salary + size\_year\_of\_joining + size\_email

max\_num\_users equ 100

size\_users\_array equ size\_user\_record\*max\_num\_users

current\_number\_of\_users dq 0

current\_number\_of\_badgers dq 0

current\_year equ 2024

current\_mon equ 1

prnt\_enter\_badger\_id db "Enter Badgers ID & enter the numbers only without 'b' i.e 6 digits only:", 10, 0

prnt\_enter\_badger\_name db "Enter Badger name:", 10, 0

home\_sett db 10,\

"Select the number that corresponds with the Home setting of the Badger", 10,\

" 1 Settfield", 10,\

" 2 Badgerton", 10,\

" 3 Stripeville", 10, \

"Please Enter Option 1 - 3", 10, 0

prnt\_enter\_badger\_mass db "Enter Badger mass to the nearest whole KG :", 10, 0

prnt\_enter\_badger\_stripes db "Enter number of stripes b/w 0-255:", 10, 0

prnt\_enter\_badger\_sex db "Enter Badger sex:", 10, \

" 1 Male", 10, \

" 2 Female", 10, 0

prnt\_enter\_badger\_birth\_month db "Enter the birth month of the Badger, ranging (0-11) where ", 10, 0

prnt\_enter\_badger\_birth\_year db "Enter the year Badger was born:", 10, 0

prnt\_number\_of\_badgers db "Number of Badgers: ", 0

prnt\_badger\_birth\_format db "number 0 represents January, 1 is February, ... and 11 is December Default is January, if nothing is entered.", 0

prnt\_enter\_badger\_keeper db "Please enter the keeper ID of the badger (7 digit number)", 10, 0

prnt\_badger\_add\_success db "A new Badger is added successfully!!", 10, 0

settfield db "Settfield", 0

badgerton db "Badgerton", 0

stripeville db "Stripeville", 0

male db "Male", 0

female db "Female", 0

prnt\_del\_badg\_id\_notfound db "No badger with this id was found", 10, 0

prnt\_badg\_array\_empty db "The badger array is empty", 10, 0

prnt\_\_badger\_letter\_b db "b", 0

prnt\_\_badger\_ID db "Badger ID: ", 0

prnt\_\_badger\_name db "Full Name: ", 0

prnt\_\_badger\_home\_sett db "Home Sett: ", 0

prnt\_\_badger\_mass db "Mass: ", 0

prnt\_\_badger\_stripes db "No of Stripes: ", 0

prnt\_\_badger\_stripiness db "Stripiness: ", 0

prnt\_\_badger\_sex db "Sex: ", 0

prnt\_\_badger\_birth\_month db "Badger birth month: ", 0

prnt\_\_badger\_birth\_year db "Badger birth year: ", 0

prnt\_\_badger\_letter db "b", 0

prnt\_\_badger\_birth db "Badger DOB: ", 0

prnt\_\_badger\_age db "Age: ", 0

prnt\_\_badger\_keeper db "Keeper: ", 0

prnt\_prompt\_badg\_delete\_id db "Please enter badger ID you want to delete", 0

prnt\_badg\_id\_err db "No badger with this ID exists", 0

prnt\_del\_badg\_id\_found db "Badger found", 0

size\_badger\_id equ 4

size\_badger\_name equ 64

size\_badger\_home\_sett equ 12

size\_badger\_mass equ 2

size\_badger\_stripes equ 1

size\_badger\_sex equ 7

size\_badger\_birth\_month equ 1

size\_badger\_birth\_year equ 2

size\_badger\_record equ size\_delete + size\_badger\_id + size\_badger\_name + size\_badger\_home\_sett + size\_badger\_mass + size\_badger\_stripes + size\_badger\_sex + size\_badger\_birth\_month + size\_badger\_birth\_year + size\_id

max\_num\_badgers equ 500

size\_badgers\_array equ size\_badger\_record \* max\_num\_badgers; Note that max\_num\_badgers is an immediate operand since it is defined at build-time

section .bss

users: resb size\_users\_array

temp\_users: resb size\_user\_record

badgers: resb size\_badgers\_array

temp\_badgers: resb size\_badger\_record

section .text

add\_user:

push rbx

push rcx

push rdx

push rdi

push rsi

mov rcx, temp\_users

mov byte[rcx], 1

inc rcx

; get user id

mov rdi, prnt\_enter\_id

call print\_string\_new ; print message

call read\_uint\_new

mov DWORD[rcx], eax

cmp rax, 999999

jle .error

cmp rax, 10000000

jge .error

add rcx, size\_id

mov rdi, prnt\_enter\_forename

call print\_string\_new ; print message

call read\_string\_new

mov rbx, rax

call string\_length

cmp rax, 63

ja .error

mov rsi, rbx ; store address in rsi

mov rdi, rcx

call copy\_string

add rcx, size\_forename

mov rdi, prnt\_enter\_surname

call print\_string\_new ; print message

call read\_string\_new

mov rbx, rax

call string\_length

cmp rax, 63 ; check if string is larger than 63 characters, allowing space for null

ja .error

mov rsi, rbx ; store address in rsi

mov rdi, rcx

call copy\_string

add rcx, size\_surname

mov rdi, prnt\_enter\_year

call print\_string\_new ; print message

call read\_uint\_new

mov WORD[rcx], ax ; we are only going to copy 1 word of RAX

add rcx, size\_year\_of\_joining

mov rdi, prnt\_enter\_salary

call print\_string\_new ; print message

call read\_uint\_new

mov DWORD[rcx], eax

;get department

mov rdi, department\_question

call print\_string\_new ; print message

call read\_int\_new

cmp rax, 1

jne .check\_gift\_shop

mov rsi, park\_keeper

jmp .add\_department

.check\_gift\_shop:

cmp rax, 2

jne .check\_cafe

mov rsi, gift\_shop

jmp .add\_department

.check\_cafe:

cmp rax, 3

jne .department\_error

mov rsi, cafe

jmp .add\_department

.department\_error:

mov rax, -2

jmp .error

.add\_department:

add rcx, size\_salary

mov rdi, rcx

call copy\_string ; copy string from into user record in array

; get email

mov rdi, prnt\_enter\_email

call print\_string\_new ; print message

call read\_string\_new

mov rbx, rax

call string\_length

cmp rax, 63 ; check if string is larger than 63 characters, allowing space for null

ja .error

mov rsi, rbx

lea rsi, [rbx + rax - 10] ; go the last 10 bytes of the user string and check its "@jnz.co.uk"

lea rdi, [email]

call strings\_are\_equal

cmp rax, 1

jne .error

add rcx, size\_dept

mov rsi, rbx

mov rdi, rcx ; store address in rsi

call copy\_string

call print\_nl\_new

mov rdi, prnt\_user\_success ; inform user is added successfully!!

call print\_string\_new

call print\_nl\_new

; copy temp array into array

mov rcx, users ; base address of users array

mov rsi, temp\_users

mov rbx, 0

; find an empty spot to copy to

.find\_spot\_to\_add\_user:

cmp byte[rcx],1

jne .do\_user\_copy

add rcx, size\_user\_record

inc rbx

cmp rbx, max\_num\_users

jl .find\_spot\_to\_add\_user

.no\_space\_user:

mov rdi, prnt\_array\_full

call print\_string\_new

.do\_user\_copy:

mov rdi, rcx

call copy\_string

add rsi, size\_delete ; delete flag

lea rdi, [rcx + size\_delete]

call copy\_string

add rsi, size\_id ; size ID

lea rdi, [rcx + size\_delete + size\_id]

call copy\_string

add rsi, size\_forename ; forename

lea rdi, [rcx + size\_delete + size\_id + size\_forename]

call copy\_string

add rsi, size\_surname ; surname

lea rdi, [rcx + size\_delete + size\_forename+size\_id+size\_surname]

call copy\_string

add rsi, size\_year\_of\_joining ; year of joining

lea rdi, [rcx + size\_delete + size\_forename+size\_id+size\_surname+size\_year\_of\_joining]

call copy\_string

add rsi, size\_salary ; salary

lea rdi, [rcx + size\_delete + size\_forename+size\_id+size\_surname+size\_year\_of\_joining+size\_salary]

call copy\_string

add rsi, size\_dept ; department

lea rdi, [rcx + size\_delete + size\_forename+size\_id+size\_surname+size\_year\_of\_joining+size\_salary+size\_dept]

call copy\_string

inc qword[current\_number\_of\_users] ; increment our number of users counter, since we have just added a record into the array.

mov rdi, temp\_users ; clear temp array here

mov rcx, size\_user\_record

xor al, al

rep stosb

pop rsi

pop rdi

pop rdx

pop rcx

pop rbx

ret

.error:

mov rdi, temp\_users

mov rcx, size\_user\_record

xor al, al

rep stosb

mov rdi, prnt\_error

call print\_string\_new

pop rsi

pop rdi

pop rdx

pop rcx

pop rbx

ret ; End function add\_user

list\_all\_users:

push rbx

push rcx

push rdx

push rdi

push rsi

lea rsi, [users] ; load base address of the users array into RSI

mov rcx, 0 ; we will use RCX for the counter in our loop

.start\_loop: ; this is the start of our loop

mov rdi, rsi

cmp BYTE[rdi], 1 ; put a check here for the delete flag

jne .goto\_next\_user

mov rdi, prnt\_\_staff ; "Staff ID: "

call print\_string\_new

mov rdi, prnt\_\_staff\_letter\_p ; prints the "p"

call print\_string\_new

mov edi, [rsi + size\_delete] ; the staff ID

call print\_uint\_new

call print\_nl\_new

mov rdi, prnt\_\_name ; "Name: "

call print\_string\_new

lea rdi, [rsi + size\_delete + size\_id ] ; Forname

call print\_string\_new

mov rdi,' ' ; space character, between forename and surname.

call print\_char\_new ; print a space

lea rdi, [rsi + size\_delete + size\_id + size\_forename] ; surname

call print\_string\_new

call print\_nl\_new

mov rdi, prnt\_\_year ; Year of Joining:

call print\_string\_new

movzx rdi, WORD[rsi + size\_delete + size\_id+size\_forename + size\_surname] ; year

mov rbx, rdi

call print\_uint\_new

call print\_nl\_new

mov rdi, prnt\_\_salary ; Salary

call print\_string\_new

mov edi, DWORD[rsi + size\_delete + size\_id+size\_forename+size\_surname+ size\_year\_of\_joining]

call print\_uint\_new

mov rdi, prnt\_\_staff\_salary\_gbp ; prints the gbp as currency

call print\_string\_new

call print\_nl\_new

mov rdi, prnt\_\_staff\_current\_salary ; current salary after adding bonus

call print\_string\_new

movzx rdi, WORD[rsi + size\_delete + size\_id + size\_forename + size\_surname] ; start year

mov rax, current\_year

sub rax, rdi ; years\_worked = current\_year - start\_year

imul rax, 200 ; bonus = years\_worked \* 200

mov edi, DWORD[rsi + size\_delete + size\_id+size\_forename+size\_surname+ size\_year\_of\_joining] ; starting salary

add rdi, rax ; current\_salary = starting\_salary + bonus

call print\_uint\_new

mov rdi, prnt\_\_staff\_salary\_gbp ; prints gbp as currency

call print\_string\_new

call print\_nl\_new

mov rdi, prnt\_\_dept ; Department:

call print\_string\_new

lea rdi, [rsi + size\_delete + size\_id+size\_forename+size\_surname+size\_year\_of\_joining + size\_salary] ;

call print\_string\_new

call print\_nl\_new

mov rdi, prnt\_\_email ; Email

call print\_string\_new

lea rdi, [rsi + size\_delete + size\_id+size\_forename+size\_surname+size\_year\_of\_joining+size\_salary+size\_dept]

call print\_string\_new

call print\_nl\_new

call print\_nl\_new ; go to next user record

.goto\_next\_user:

add rsi, size\_user\_record ; move the address to point to the next record in the array

add rcx, size\_user\_record ; increment our counter variable

cmp rcx, size\_users\_array

jl .start\_loop ; jump back to the start of the loop (unconditional jump)

.finish\_loop:

pop rsi

pop rdi

pop rdx

pop rcx

pop rbx

ret

prnt\_\_number\_of\_users:

push rdi

mov rdi, prnt\_number\_of\_users

call print\_string\_new

mov rdi, [current\_number\_of\_users]

call print\_uint\_new

call print\_nl\_new

pop rdi

ret ; End function prnt\_\_number\_of\_users

prnt\_\_main\_menu:

push rdi

mov rdi, prnt\_main\_menu

call print\_string\_new

pop rdi

ret ; End function prnt\_\_main\_menu

string\_length: ;common function for both badgers and staff for length

push rcx

push rbx

push rdi

sub rcx,rcx ; this sets the max size to look for to be

not rcx

mov al, 0 ; We want to look for the byte 0, null terminator

mov rdi, rbx ; set the start of the string

cld

repne scasb ; perform search operation

sub rdi, rbx ; perform search operation

dec rdi

mov rax, rdi

pop rdi

pop rbx

pop rcx

ret

add\_badgers:

push rbx

push rcx

push rdx

push rdi

push rsi

mov rcx, temp\_badgers

mov byte[rcx], 1

inc rcx

mov rdi, prnt\_enter\_badger\_id ; get badger id

call print\_string\_new

call read\_uint\_new

mov DWORD[rcx], eax

cmp rax, 99999

jle .badger\_error

cmp rax, 1000000

jge .badger\_error

add rcx, size\_badger\_id

mov rdi, prnt\_enter\_badger\_name ; get badger name

call print\_string\_new

call read\_string\_new

mov rbx, rax

call string\_length

cmp rax, 63

ja .badger\_error

mov rsi, rbx ; store address in rsi

mov rdi, rcx

call copy\_string

mov rdi, home\_sett

call print\_string\_new

call read\_int\_new

cmp rax, 1

jne .check\_badgerton

mov rsi, settfield

jmp .add\_home\_sett

.check\_badgerton:

cmp rax, 2

jne .check\_stripeville

mov rsi, badgerton

jmp .add\_home\_sett

.check\_stripeville:

cmp rax, 3

jne .home\_sett\_error

mov rsi, stripeville

jmp .add\_home\_sett

.home\_sett\_error:

mov rax, -2

jmp .badger\_error

.add\_home\_sett:

add rcx, size\_badger\_name

mov rdi, rcx

call copy\_string

;get badger mass

add rcx, size\_badger\_home\_sett

mov rdi, prnt\_enter\_badger\_mass

call print\_string\_new

call read\_uint\_new

mov WORD[rcx], ax

;get Badger stripes

add rcx, size\_badger\_mass

mov rdi, prnt\_enter\_badger\_stripes

call print\_string\_new

call read\_uint\_new

mov BYTE[rcx], al

;get Badger sex

mov rdi, prnt\_enter\_badger\_sex

call print\_string\_new ; print message

call read\_int\_new ;read in option

cmp rax, 1

jne .check\_female

mov rsi, male

jmp .add\_badger\_sex

.check\_female:

cmp rax, 2

jne .badger\_sex\_error

mov rsi, female

jmp .add\_badger\_sex

.badger\_sex\_error:

mov rax, -2

jmp .badger\_error

.add\_badger\_sex:

add rcx, size\_badger\_stripes

mov rdi, rcx

call copy\_string

;get Badger birth month

add rcx, size\_badger\_sex

mov rdi, prnt\_enter\_badger\_birth\_month

call print\_string\_new

mov rdi, prnt\_badger\_birth\_format

call print\_string\_new

call print\_nl\_new

call read\_uint\_new

mov BYTE[rcx], al

cmp rax, 12

jge .badger\_error

add rcx, size\_badger\_birth\_month

;get Badger birth year

mov rdi, prnt\_enter\_badger\_birth\_year

call print\_string\_new

call read\_uint\_new

mov WORD[rcx], ax

cmp rax, current\_year

jg .badger\_error ; badger birth year should not be greater than current year

call print\_nl\_new

add rcx, size\_badger\_birth\_year

; add badger keeper

mov rdi, prnt\_enter\_badger\_keeper

call print\_string\_new

call read\_uint\_new

; use the same check/comparison as keeper id

mov DWORD[rcx], eax ; we are only going to copy the least significant byte of RAX (AL),

cmp rax, 999999

jle .badger\_error

cmp rax, 10000000

jge .badger\_error

; all details added

mov rdi, prnt\_badger\_add\_success

call print\_string\_new

call print\_nl\_new

mov rcx, badgers ; base address of badgers array

mov rsi, temp\_badgers ; address of temp\_badgers

mov rbx, 0

.find\_spot\_to\_add\_badger:

cmp byte[rcx], 1

jne .do\_badger\_copy

add rcx, size\_badger\_record

inc rbx

cmp rbx, max\_num\_badgers

jl .find\_spot\_to\_add\_badger

.no\_space\_badger:

mov rdi,prnt\_array\_full

call print\_string\_new

.do\_badger\_copy:

mov rdi, rcx

call copy\_string ; copy delete flag

add rsi, size\_delete

lea rdi, [rcx + size\_delete]

call copy\_string ; copy id

add rsi, size\_badger\_id

lea rdi, [rcx + size\_delete + size\_badger\_id]

call copy\_string ; name

add rsi, size\_badger\_name

lea rdi, [rcx + size\_delete + size\_badger\_name+size\_badger\_id]

call copy\_string ; sett

add rsi, size\_badger\_home\_sett

lea rdi, [rcx + size\_delete + size\_badger\_name+size\_badger\_id+size\_badger\_home\_sett]

call copy\_string ; mass

add rsi, size\_badger\_mass

lea rdi, [rcx + size\_delete + size\_badger\_name+size\_badger\_id+size\_badger\_home\_sett+size\_badger\_mass]

call copy\_string ; stripes

add rsi, size\_badger\_stripes

lea rdi, [rcx + size\_delete + size\_badger\_name+size\_badger\_id+size\_badger\_home\_sett+size\_badger\_mass+size\_badger\_stripes]

call copy\_string ; sex

add rsi, size\_badger\_sex

lea rdi, [rcx + size\_delete + size\_badger\_name+size\_badger\_id+size\_badger\_home\_sett+size\_badger\_mass+size\_badger\_stripes+size\_badger\_sex]

call copy\_string ; birth month

add rsi, size\_badger\_birth\_month

lea rdi, [rcx + size\_delete + size\_badger\_name+size\_badger\_id+size\_badger\_home\_sett+size\_badger\_mass+size\_badger\_stripes+size\_badger\_sex+size\_badger\_birth\_month]

call copy\_string ; birth year

add rsi, size\_badger\_birth\_year

lea rdi, [rcx + size\_delete + size\_badger\_name+size\_badger\_id+size\_badger\_home\_sett+size\_badger\_mass+size\_badger\_stripes+size\_badger\_sex+size\_badger\_birth\_month + size\_badger\_birth\_year]

call copy\_string ; birth year

inc QWORD[current\_number\_of\_badgers] ; increment our number of users counter, since we have just added a record into the array.

jmp .end\_add\_badg

.badger\_error:

mov rdi, prnt\_error

call print\_string\_new

call print\_nl\_new

.end\_add\_badg:

; clear temp array here

mov rdi, temp\_badgers

mov rcx, size\_badger\_record

xor al, al

rep stosb

pop rsi

pop rdi

pop rdx

pop rcx

pop rbx

ret ; End function add\_user

list\_all\_badgers:

; Takes no parameters (badgers is global)

; Lists full details of all badgers in the array

push rbx

push rcx

push rdx

push rdi

push rsi

lea rsi, [badgers] ; load base address of the badgers array into RSI In other words, RSI points to the badgers array.

mov rcx, 0

.start\_badger\_loop: ; this is the start of our loop

mov rdi, rsi ; put the pointer to the current record in RDI, to pass to the print\_string\_new function

cmp byte[rdi], 1 ; check badger delete flag

jne .goto\_next\_badger

mov rdi, prnt\_\_badger\_ID ; Badger ID:

call print\_string\_new

mov rdi, prnt\_\_badger\_letter\_b ; the letter b

call print\_string\_new

mov edi, [rsi + size\_delete] ; offset of badger ID

call print\_uint\_new

call print\_nl\_new

mov rdi, prnt\_\_badger\_name ; display Badger name

call print\_string\_new

lea rdi, [rsi + size\_delete + size\_badger\_id] ; offset of badger Name

call print\_string\_new

call print\_nl\_new

mov rdi, prnt\_\_badger\_home\_sett ; display Badger Sett

call print\_string\_new

lea rdi, [rsi + size\_delete + size\_badger\_id + size\_badger\_name] ; sett

call print\_string\_new

call print\_nl\_new

;display Badger mass

mov rdi, prnt\_\_badger\_mass

call print\_string\_new

movzx rdi, WORD[rsi + size\_delete + size\_badger\_id+size\_badger\_name+size\_badger\_home\_sett] ; mass

call print\_uint\_new ; print the mass

call print\_nl\_new

;display badger stripes

mov rdi, prnt\_\_badger\_stripes

call print\_string\_new

movzx rdi, BYTE[rsi + size\_delete + size\_badger\_id+size\_badger\_name+size\_badger\_home\_sett+size\_badger\_mass] ; stripes

call print\_uint\_new

call print\_nl\_new

; display stripiness

mov rdi, prnt\_\_badger\_stripiness

call print\_string\_new

movzx rax, BYTE[rsi + size\_delete + size\_badger\_id+size\_badger\_name+size\_badger\_home\_sett+size\_badger\_mass] ; stripes

movzx rbx, WORD[rsi + size\_delete + size\_badger\_id+size\_badger\_name+size\_badger\_home\_sett] ; mass

mul rbx ; stripiness = mass \* stripes

mov rdi, rax

call print\_uint\_new

call print\_nl\_new

; display badger sex

mov rdi, prnt\_\_badger\_sex

call print\_string\_new

lea rdi, [rsi + size\_delete + size\_badger\_id+size\_badger\_name+size\_badger\_home\_sett+size\_badger\_mass+ size\_badger\_stripes]; move the pointer along by 129 bytes from the base address of the record (combined size of the forename and surname strings, and age)

call print\_string\_new

call print\_nl\_new

; display birth year & month

mov rdi, prnt\_\_badger\_birth

call print\_string\_new

movzx rdi, BYTE[rsi + size\_delete + size\_badger\_id+size\_badger\_name+size\_badger\_home\_sett+size\_badger\_mass+size\_badger\_stripes+size\_badger\_sex] ; month

call print\_uint\_new

mov rdi, '/'

call print\_char\_new

movzx rdi, WORD[rsi + size\_delete + size\_badger\_id+size\_badger\_name+size\_badger\_home\_sett+size\_badger\_mass+size\_badger\_stripes+size\_badger\_sex+size\_badger\_birth\_month] ; year

call print\_uint\_new

call print\_nl\_new

; display badger current age

mov rdi,prnt\_\_badger\_age

call print\_string\_new

movzx rdi, WORD[rsi + size\_delete + size\_badger\_id+size\_badger\_name+size\_badger\_home\_sett+size\_badger\_mass+size\_badger\_stripes+size\_badger\_sex+size\_badger\_birth\_month] ; year

mov rax, current\_year

sub rax, rdi ; current\_year - badger\_birth\_year = gives the badger\_age

movzx rdi, BYTE[rsi + size\_delete + size\_badger\_id+size\_badger\_name+size\_badger\_home\_sett+size\_badger\_mass+size\_badger\_stripes+size\_badger\_sex] ; month

mov rbx, current\_mon

cmp rbx, rdi

jge .curr\_month\_greater\_than\_birth ; unless current month is before their birthday,

dec rax

.curr\_month\_greater\_than\_birth:

mov rdi, rax

call print\_uint\_new

call print\_nl\_new

; assigned keeper

mov rdi, prnt\_\_badger\_keeper

call print\_string\_new

mov rdi, prnt\_\_staff\_letter\_p

call print\_string\_new

mov edi, [rsi + size\_delete + size\_badger\_id+size\_badger\_name+size\_badger\_home\_sett+size\_badger\_mass+size\_badger\_stripes+size\_badger\_sex+size\_badger\_birth\_month + size\_badger\_birth\_year]

call print\_uint\_new

call print\_nl\_new

call print\_nl\_new

.goto\_next\_badger:

add rsi, size\_badger\_record ; move the address to point to the next record in the array

add rcx, size\_badger\_record ; increase our counter variable

cmp rcx, size\_badgers\_array

jl .start\_badger\_loop ; jump back to the start of the loop (unconditional jump)

.finish\_badger\_loop:

pop rsi

pop rdi

pop rdx

pop rcx

pop rbx

ret ; End function list\_badgers

delete\_badger: ; delete badger function

push rbx

push rcx

push rdx

push rdi

push rsi

.read\_badg\_del\_id: ; get ID of badger from user

mov rdi, prnt\_prompt\_badg\_delete\_id

call print\_string\_new

call print\_nl\_new

call read\_uint\_new

cmp rax, 99999 ; value checks for a valid 6 digit number

jl .badg\_id\_err

cmp rax, 1000000

jg .badg\_id\_err

; if it doesn't jump, Badger ID is fine

; perform search operation to see if this badger id exists

mov rbx, badgers ; base address

mov rcx, 0 ; counter

.start\_find\_badg\_del\_loop:

.find\_badg\_id\_del:

cmp byte[rbx], 1

jne .goto\_next\_badg\_delete

mov edi, dword[rbx + size\_delete] ; compare badger ID with user input in rax

cmp rax, rdi

je .badg\_del\_id\_found

.goto\_next\_badg\_delete: ; go to the next address where a record should be

add rbx, size\_badger\_record

inc rcx

.do\_stop\_badg\_loop: ; check if or if not youǘe gone over

cmp rcx, max\_num\_badgers

jg .badg\_id\_err

jmp .find\_badg\_id\_del

.end\_find\_badg\_del\_loop:

.badg\_del\_id\_found: ; record exists with badger ID

mov rdi, prnt\_del\_badg\_id\_found

call print\_string\_new

call print\_nl\_new

mov byte[rbx], 0 ; set the delete flag to 0

dec qword[current\_number\_of\_badgers] ; decrease the counter

jmp .end\_badg\_del

.badg\_id\_err: ; record doesn't exist

mov rdi, prnt\_del\_badg\_id\_notfound

call print\_string\_new

call print\_nl\_new

.end\_badg\_del:

pop rsi

pop rdi

pop rdx

pop rcx

pop rbx

ret

delete\_user: ; delete user record

push rbx

push rcx

push rdx

push rdi

push rsi

.read\_user\_del\_id:

mov rdi, prnt\_prompt\_user\_delete\_id

call print\_string\_new

call print\_nl\_new

call read\_uint\_new

cmp rax, 999999

jle .error\_del

cmp rax, 10000000

jge .error\_del

mov rbx, users ; base address of user array

mov rcx, 0 ; counter

.find\_id\_del\_loop:

.find\_user\_id:

cmp byte[rbx], 1 ; delete flag must be set to 1

jne .goto\_next\_user\_delete

mov edi, dword[rbx + size\_delete]

cmp rax, rdi

je .found\_user\_now\_delete

jmp .goto\_next\_user\_delete

.goto\_next\_user\_delete:

add rbx, size\_user\_record

inc rcx

.check\_stop\_user\_loop:

cmp rcx, max\_num\_users

jg .error\_del

jmp .find\_user\_id

.end\_id\_del\_loop:

.found\_user\_now\_delete:

mov rdi, prnt\_del\_user\_id\_found

call print\_string\_new

call print\_nl\_new

mov byte[rbx], 0 ; set the byte to 0

dec QWORD[current\_number\_of\_users]

jmp .end\_del

.error\_del:

mov rdi, prnt\_del\_user\_id\_notfound

call print\_string\_new

call print\_nl\_new

.end\_del:

pop rsi

pop rdi

pop rdx

pop rcx

pop rbx

ret

prnt\_\_number\_of\_badgers:

; No parameters

; Displays number of users in list (to STDOUT)

push rdi

mov rdi, prnt\_number\_of\_badgers

call print\_string\_new

mov rdi, [current\_number\_of\_badgers]

call print\_uint\_new

call print\_nl\_new

pop rdi

ret ; End function prnt\_\_number\_of\_badgers

main:

mov rbp, rsp ; for correct debugging

push rbp

sub rsp, 32

mov rdi, prnt\_badg\_art ; Assuming prnt\_badg\_art is defined earlier

call print\_string\_new

call print\_nl\_new

; Take input for the current month and set it as the new current month

;mov rdi, prnt\_current\_month

;call print\_string\_new

;call read\_int\_new

;mov [current\_mon], rax ; Set the entered value as the new current month

; Take input for the current year and set it as the new current year

;mov rdi, prnt\_current\_year

;call print\_string\_new

;call read\_int\_new

;mov [current\_year], rax ; Set the entered value as the new current year

.menu\_loop:

call prnt\_\_main\_menu

call read\_int\_new ; menu option (number) is in RAX

mov rdx, rax

mov rdi, prnt\_option\_selected

call print\_string\_new

mov rdi, rdx

call print\_int\_new

call print\_nl\_new

; Now jump to the correct option

cmp rdx, 1

je .option\_1

cmp rdx, 2

je .option\_2

cmp rdx, 3

je .option\_3

cmp rdx, 4

je .option\_4

cmp rdx, 5

je .option\_5

cmp rdx, 6

je .option\_6

cmp rdx, 7

je .option\_7

cmp rdx, 8

je .option\_8

cmp rdx, 9

je .option\_9

;Display error and loop back to input option in case of any error

mov rdi, prnt\_invalid\_option

call print\_string\_new

jmp .menu\_loop

.option\_1: ; 1 Add User

mov rdx, [current\_number\_of\_users] ; This is indirect, hence [] to dereference

cmp rdx, max\_num\_users ; Note that max\_num\_users is an immediate operand since it is defined at build-time

jl .array\_is\_not\_full ; If current\_number\_of\_users < max\_num\_users then array is not full, so add new user.

mov rdi, prnt\_array\_full ; display "array is full" message and loop back to main menu

call print\_string\_new

jmp .menu\_loop

.array\_is\_not\_full:

call print\_nl\_new

call add\_user

jmp .menu\_loop

.option\_2: ; 2 List All Users

call print\_nl\_new

call list\_all\_users

jmp .menu\_loop

.option\_3: ; 3 Count Users

call print\_nl\_new

call prnt\_\_number\_of\_users

jmp .menu\_loop

.option\_4: ; 4 Delete user

call print\_nl\_new

mov rdx, current\_number\_of\_users

cmp rdx, 0

jg .staff\_array\_isnt\_empty ; If current\_number\_of\_users <= 0 then array is empty

mov rdi, prnt\_array\_empty ; display array is empty message and loop back to main menu

call print\_string\_new

jmp .menu\_loop

.staff\_array\_isnt\_empty:

call print\_nl\_new

call delete\_user

jmp .menu\_loop

.option\_5: ; 5 Add Badgers

mov rdx, [current\_number\_of\_badgers] ; This is indirect, hence [] to dereference

cmp rdx, max\_num\_badgers ; Note that max\_num\_badgers is an immediate operand since it is defined at build-time

jl .badger\_array\_is\_not\_full ; If current\_number\_of\_badgers < max\_num\_badgers then array is not full, so don't add new badger

mov rdi, prnt\_array\_full ; display "array is full" message and loop back to main menu

call print\_string\_new

jmp .menu\_loop

.badger\_array\_is\_not\_full:

call add\_badgers

jmp .menu\_loop

.option\_6: ; 6 List All Badgers

call print\_nl\_new

call list\_all\_badgers

jmp .menu\_loop

.option\_7: ; 7 Count Badgers

call print\_nl\_new

call prnt\_\_number\_of\_badgers

jmp .menu\_loop

.option\_8:

mov rdx, [current\_number\_of\_badgers]

cmp rdx, 0

jg .badger\_array\_is\_not\_empty

mov rdi, prnt\_badg\_array\_empty

call print\_string\_new

jmp .menu\_loop

.badger\_array\_is\_not\_empty:

call delete\_badger

jmp .menu\_loop

.option\_9: ; 9 Exit

mov rdi, prnt\_program\_exit

call print\_string\_new

xor rax, rax

add rsp, 32

pop rbp

ret