COMP7023 Coursework 1

1 Introduction

The purpose of this coursework is to develop software using x86 64-bit assembly. This software is an archive system to keep track of the badgers and staff in the zoo. The program was written with the help of Joe Norton library. This library presents different function implementations of the essential operation as print string or read string. In Table 1 there is a brief explanation of these functions.

Function	Input register	Return register	Explanation
print string new	RDI	RAX	Print string in the command line
print int new	RDI	RAX	Print signed int in the command line
print uint new	RDI	RAX	Print unsigned int in the command line
print nl new		RAX	Jump to the consecutive line, printing 0x0A the newline char
print char new	RDI	RAX	Print a char in the command line
read_char new		RAX	Read a char from the command line and return the char in RAX.
read_string new		RAX	Read a string until the linefeed. The null terminator replaces linefeed. Rax register returns the value.
atoi	RDI	RAX	Convert a string pointed to by RDI to a 64-bit integer. RAX return the converted number or -1 if an error occurs.
read_uint new		RAX	Read an unsigned 64-bit integer value from the console. RAX return the converted number or -1 if an error occurs.
read_int new			Read a signed 64-bit integer value from the console. RAX return the converted number or -1 if an error occurs.
copy string	RDI RSI	RSI	The function copies the string in RDI to the RSI register.
strings are equal	RDI RSI	RAX	The function compares the string in RDI and RSI. If they are equal, return RAX=1 else, return 0

Table 1: Library functions

2 Register 64-bits

To understand how 64-bit technology works, the report needs to focus the reader's attention on the structure of 64-bit registers. The lower 32 bits, 16 bits, and 8 bits of each register are directly accessible with a sub-register. In this way, the developer has more flexibility and control of the memory [1]. All the operation with a sub-register 32-bits is zero-extended to the entire 8-bytes, but the sub-register 16-bits and 8-bits are not. Additionally we can access to the lower bytes using the operands Byte,Word and Dword.

Size (in Bits)			
64	32	16	8
RAX	EAX	AX	AH/AL
RBX	EBX	BX	BH/BL
RCX	ECX	CX	CH/CL
RDX	EDX	DX	DH/DL
RDI	EDI	DI	DIL
RSI	ESI	SI	SIL
RBP	EBP	BP	BPL
RSP	ESP	SP	SPL
R8∼R15	R8D∼R15D	R8W∼R15W	R8L∼R15L

Figure 1 64 bits register structure [2]

2.1 The Setup

The IDE used to develop, debug, and test the software is SASM. SASM (SimpleASM) is a simple cross-platform Open-Source IDE for the NASM, MASM, GAS, and FASM assembly languages [3]. SASM is installed on a Lubuntu virtual machine.

3 Memory considerations

3.1 Badger

Each one badger entry must contain all these following parameters.

- Badger ID
- Name
- Name Home sett (can be any one of Settfield, Badgerton, or Stripeville)
- Number of stripes (in the range from 0 to 255)
- Sex (M or F)
- Month of birth
- Year of birth
- Staff ID

Each field must respect the system requirements. Badger ID is a string containing b as the first character followed by six digits and a null terminator.

The name should be less than 64 characters, and the home sett less than 12. Furthermore, Mass, stripes, Sex and Month of birth occupy 1 byte each and are represented by an unsigned int 8 bits. Instead, the birth year must contain four digits, meaning 2 bytes. The decision to use 2 bytes is because, with only 1 byte (8 bits), there are only 256 different permutations (i.e., combinations of zero and one), and we can only have 256 possible values (i.e., 2^8). So, to represent a year with four digits, we need 2 bytes (2^{16} possible values). This value is an unsigned integer 16-bit, and the program will access the lower 16-bit of the register to store it. The last parameter is Staff id, a string which starts with the letter p followed by seven digits plus and null terminator. The decision to use string and not a 32-bit integer to store staff id and badger id is because integers surely decrement memory usage but can present various other issues such as integer overflow or conversion problems with the first character. Considering that the badger array can store a maximum of 500 badgers and one badger is 99 bytes, as shown in Figure 2, the array size will be 49.5 Kbyte.

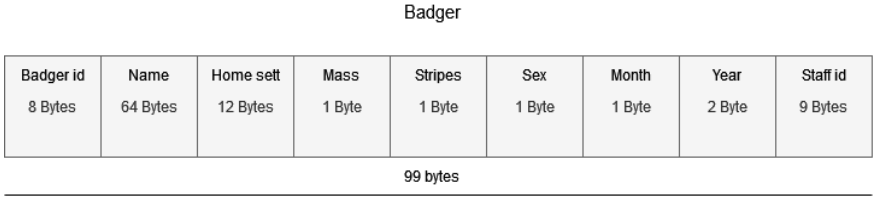

Figure 2: Badger memory representation

3.2 Staff

Each staff entry must contain all the following parameters.

- Surname
- First Name
- Staff Id
- Department (can be any one of Park Keeper, Gift Shop or Cafe)
- Starting annual salary in £ (whole £ only)
- Year of joining
- Email address

A Staff member contains a Staff id that is a string of 9 characters (9 bytes), and it must contain the letter p as the first character followed by seven digits and the null terminator. Surname, first name and email address are all strings of 64 characters, including the null terminator. The year of joining is a four-digit number represented by a UINT16. The department is a static string of 12 bytes, including a null terminator, and the salary is a UINT32 in the range of in the range [0 to 232]. Considering the staff array can store a maximum of 100 staff members and one staff member is 219 bytes, as shown in Figure 3, the array size will be 21,9 Kbyte.

STAFF

Staff id	Surname	Name	Email	Year of join	Department	Salary
9 Bytes	64 Bytes	64 Bytes	64 Bytes	2 bytes	12 Byte	4 Byte

219 bytes

Figure 3: Staff memory representation

4 Testing

The design of this software is straightforward and understandable. During the development of the archive, the idea was to create functions that were not dependent on each other and to follow the Single-responsibility Principle (SRP) to have an easy way to find the error and fix them and also to give modularity to the code. Every input user was controlled with different procedures to avoid various problems, such as exceeding the variable length or not respecting the system requirements. All functions were tested individually. The software contains eight main parts that permit a user to add/delete/list or search a badger or staff member given the id. The software uses pointers in order not to create copies of existing objects in the memory. When dealing with memory below 64bit, using a pointer that correctly references the correct memory address is really usefull. Dereferencing this sort of pointer necessitates particular caution and attention, since erroneous deference might result in unanticipated outcomes or even system failures. An overview of the structure of the software is in the next image created by the disassembler cutter representing the global callgraph 4.

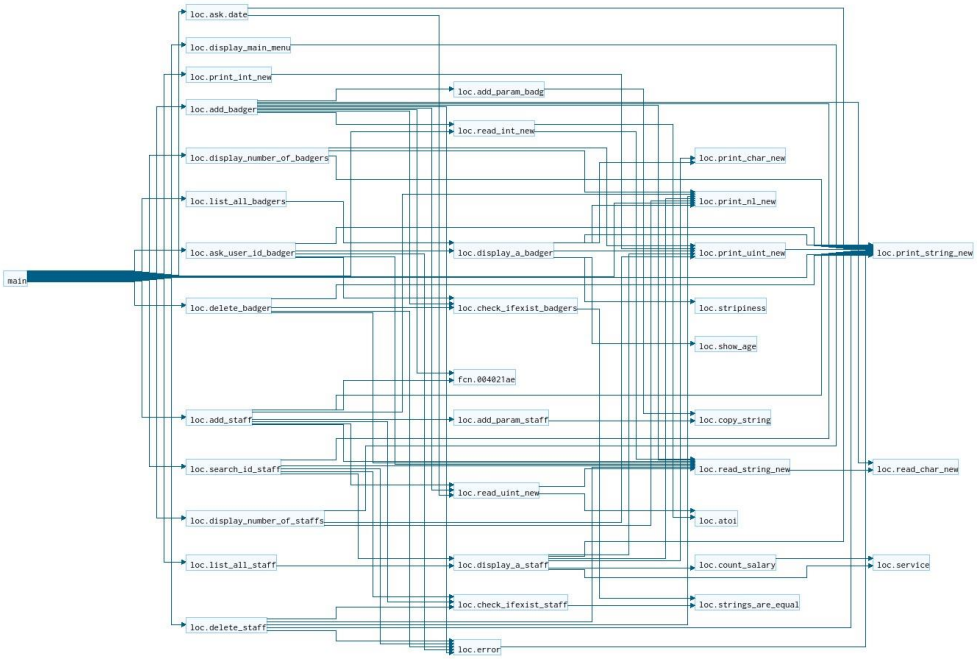

Figure 4: global callgraph

4.1 Menu and get the current date

As requested, the program must ask the current date as first operation. The date is stored in 2 two variables, one for the current year and one for the month. All the controls on the user input are performed in order to have a year formed with four digits and a month in the range of 1-12, as shown in Figure 5. If the program receives from the user a wrong input, it will ask again the date to the user. After this operation, the Menu will appear, and the user can insert a number between 1-9 and choose the operation.

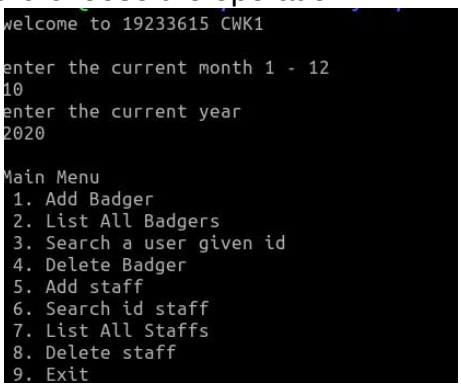

Figure 5: Main menu` and get current date

4.2 Add Badger

Add badger is the first function of the main menu; it allows to insert a badger in the array asking the user to insert all the parameters. Also all the parameters will be controlled before being allocated in the array. A badger id is a unique identifier, so a function to control this particular field was developed. An image of the execution of this function is shown in Figure 6.

Figure 6: Add badger

4.3 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. When the function reveals the parameters, two more fields are shown: stripiness and age. Stripiness is calculated by multiplying the mass by stripes, and the age of the badger is calculated by comparing the current year

and month with the year and month of birth of the badger. An image of the execution of this function is shown if Figure 7.

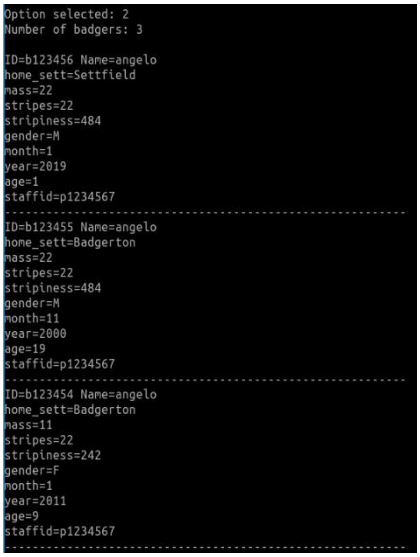

Figure 7: List all badgers

4.4 Search badger by id

This function gets a badger id as user input, searches in the array for the id, and prints the badger structure on the command line if it exits. Otherwise, an error message will be displayed, and will be asked again to the user to insert another badger id. Additionally, before calling the Search badger id function is called the software checks if the badger array is empty. In the case that the array is empty, the software jumps back to the main menu showing an error message. An image of the execution of this function is shown in Figure 8.

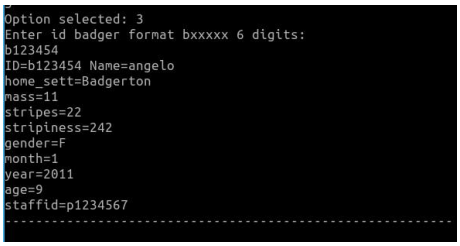

Figure 8: Search badger by id

4.5 Delete Badger

The delete badger function removes a badger with the id inserted by the user. This function shift left all badgers of one position in the register in order to overwrite the badger designed to be eliminated. If the badger is in the last position allocated of the array, the software sets all bytes of the badger as null. Also, in this case, different controls are performed on user input and before deleting the badger, the program checks if the id exists, an illustration of the execution of this function is shown in Figure 9.

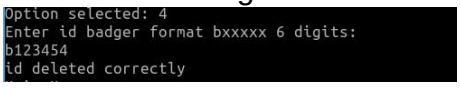

Figure 9: Delete badger

4.6 Add Staff

The Add staff function is very similar to add badger. All the controls are made on the parameters such as length and the number of digits when required. Another function checks if the id already exists; if it exists, the program asks the user to enter another id. The ending part of the email after the @ is static, so the software asks the user only the prefix of the email, and after using a function called build_email, it will concatenate the two string and insert it to the array. An illustration of the execution of this function is shown in Figure 10.

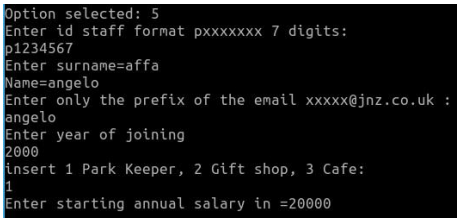

Figure 10: Add staff

4.7 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. Instead, salary is calculated by adding 200 pounds to the annual salary for each year of service, an example of the execution of this function is in Figure 11.

Figure 11: List all staff

4.8 Search staff by id

The search by staff id function required an id inserted by the user. If the id exists, the program will show all the parameters of the staff member searched; if it does not exist, an error message will show up and will be asked to the user for a new id. An illustration of the execution of this function is shown in Figure 12.

Figure 12: Search staff by id

4.9 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. Also, in this case, different controls are performed on user input and before deleting the badger, the program checks if the id exists, an illustration of the execution of this function is shown in Figure 13.

Figure 13: Delete staff

5 Conclusion

The system has been implemented in NASM according to the requirements. Overall, the program used only the necessary memory; it does not allocate memory that the program will not use. Different implementations could be performed to optimize the program. First, the program requires a way to save all the data in proper storage. The suggestion is to write all the staff and badger entries in two different files; in this way, the software has a persistent memory. Another implementation is allocating the memory dynamically to avoid any attack as buffer overflow. Additionally, a policy to protect personal data must be implemented according to the GDPR, and a security analysis of the system should be performed. In conclusion, all the requirement has been implemented, and assembly is not the best language to develop a system such as this. On the other hand, assembly assures excellent performance and is provided by massive documentation.

References

[1] Domars, *X64 architecture - windows drivers*. [Online]. Available: https://learn.microsoft. com/en-us/windows-hardware/drivers/debugger/x64-architecture.

[2] C. Pirry, H. Marco-Gisbert, and C. Begg, “A review of memory errors exploitation in x86-64,” *Computers*, vol. 9, p. 48, Jun. 2020. doi: 10.3390/computers9020048.

[3] *Sasm*. [Online]. Available: https://dman95.github.io/SASM/english.html.