ASIA PACIFIC UNIVERSITY OF TECHNOLOGY & INNOVATION

INDIVIDUAL ASSIGNMENT

Computer Systems Low Level Techniques

APD2F2206CS(CYB)

ASSIGNED DATE	:	6th December 2022
HAND IN DATE	:	16th February 2023
WEIGHTAGE	:	50%

LECTURER	:	TS. UMAPATHY EAGANATHAN
STUDENT NAME	:	CHANG SHIAU HUEI
TP NUMBER	:	TP060322

TABLE OF CONTENTS

1.0 INTRODUCTION

Assembly language is a type of programming language that directly interacts with a computer's hardware. It uses simple and understandable terms and phrases, rather than binary or hexadecimal code, making it easier for developers to work with (Preston, 2021). Assembly language can lead to increased efficiency for developers (Pedamkar, 2022) and provide more control over the computer hardware compared to higher-level programming languages, such as C or Python. Although it's not widely used, some specialized industries still utilize assembly language, such as financial firms using high-frequency trading platforms (HFT), to save processing time (Fernando, 2022).

This report will provide an in-depth examination of low-level languages, a low-level programming language that has become increasingly popular in both cybersecurity and forensic fields. The importance of assembly language, where are they commonly being used, and how low-level languages are being used in both cybersecurity and forensic fields are discussed in detail.

Furthermore, a FSEC-SS Cash Register System is coded using assembly language in this assignment. The system allows multiple users to register for FSEC-SS's events and receive the total fee for the events registered. The assignment's deliverable includes not only the coded system but also a report that details the system design and provides a user manual to guide users on how to use the system. The report covers all the necessary information that one needs to know to effectively operate the FSEC-SS Cash Register System.

2.0 RESEARCH AND ANALYSIS

2.1 - Importance of Assembly Language

Assembly language is important for a number of reasons. The first reason is its performance. Assembly language provides direct control over the hardware, allowing for the optimization of the code for a specific processor. This results in faster and more efficient code execution compared to higher-level programming languages. (Cempron et al., 2017)

Secondly, it allows for a better understanding of the processor's function and memory operation (Pal, 2017). By writing code directly in assembly, a programmer can access the registers and memory addresses directly, thereby gaining insight into how the processor retrieves values and pointers. This close interaction with the processor is particularly useful for system-level programming tasks such as compiler and device driver development. In these types of applications, it is essential to have a thorough understanding of how the processor operates in order to optimize the performance of the code. Assembly language provides a direct view into the processor's function, enabling a programmer to make the necessary modifications to improve the speed and efficiency of the code. This is especially important for system-level programming, where direct control over the hardware is critical to the success of the program.

Furthermore, assembly language is transparent and less complex compared to high-level languages, making it easier for algorithm analysis, debugging, and reducing overhead (Pal, 2017). Unlike high-level languages, which use abstract data types, assembly language has a small number of operations and is less complex, making it easier to understand and work with.

In conclusion, Assembly language is a valuable programming language that provides direct access to computer hardware and offers numerous benefits to developers. It allows for increased performance, a better understanding of the processor's function and memory operation, and offers a transparent and less complex approach to coding. These features make assembly language particularly useful for system-level programming tasks, such as compiler and device driver development. Despite its limited use in general, some specialized industries still rely on assembly language to improve the efficiency and performance of their code. Overall, Assembly language is a critical tool for programmers who require direct control over the hardware and want to achieve faster and more efficient code execution.

2.2 - Where is Assembly Language Commonly Used

Assembly language is often used in specialized industries, where the need for fast and efficient code execution is critical to the success of the application. One of the most common areas where Assembly language is used is in financial industries. Financial firms that utilize high-frequency trading platforms (HFT) are one of the biggest users of Assembly language (Fernando, 2022). These platforms use algorithms to make trades at lightning speeds (Chen, 2021), and every millisecond counts. By using Assembly language, these firms can save processing time and increase efficiency, allowing for faster trades and higher profits. Assembly language provides direct access to the computer's hardware, enabling developers to optimize the code for a specific processor and gain a competitive advantage over other firms.

Another area where Assembly language is used is in the development of system-level software. Compilers and device drivers are two examples of system-level software that uses Assembly language (Marchan, 2023; Dodos, 2021). In these applications, a thorough understanding of the processor's function is essential to the success of the program. By writing code in Assembly language, developers can access the registers and memory addresses directly, thereby gaining insight into how the processor retrieves values and pointers. This close interaction with the processor enables developers to make the necessary modifications to improve the speed and efficiency of the code.

Assembly language is also used in the development of operating systems (Gonzalez, 2016), including kernels and device drivers. These applications require close interaction with the hardware and are typically written in Assembly language for performance and efficiency reasons. Assembly language provides direct access to the processor's functions, allowing developers to fine-tune the code for specific processors and achieve optimal performance.

Finally, Assembly language is used in education and research. Many universities in Malaysia offer assembly language as part of their degree modules, for example, the Computer Systems Low Level Techniques module provided by Asia Pacific University (2023). Assembly language provides a transparent view into the computer's hardware and provides an opportunity for students and researchers to gain a deeper understanding of the workings of a computer. By writing code in Assembly language, students can learn about the underlying architecture of a computer and gain a better understanding of how high-level programming languages are translated into machine code. (Minaie & Sanati-Mehrizy, 2003)

In conclusion, Assembly language is commonly used in a number of specialized industries where fast and efficient code execution is critical to the success of the application. Financial firms utilizing high-frequency trading platforms, system-level software development, operating system development, and education and research are some of the areas where Assembly language is commonly used. Assembly language provides direct access to the computer hardware, enabling developers to optimize the code for a specific processor and achieve optimal performance. Although Assembly language is not widely used, its importance in certain industries and applications cannot be overstated.

2.3 - How Low-Level Language is Being Used in Cybersecurity and Forensic Fields

A low-level language is a type of programming language that has minimal abstraction from the underlying hardware, meaning that it closely resembles writing machine instructions (Computer Hope, 2019). Some examples include assembly language and machine code, which play an important role in the field of cybersecurity. According to Eggleston (2020), since low-level programming means that the programmer has to handle memory management, C and C++ are now also seen as low-level languages because they don't have built-in memory management. Assembly language provides a transparent view into the computer's hardware, enabling security experts to understand how data is processed and stored at the machine level. By using Assembly language, security experts can identify vulnerabilities and weaknesses in the hardware and software, and develop secure and robust security solutions to protect against cyber attacks.

One area where low-level languages are commonly used in cybersecurity is in the development of intrusion detection and prevention systems (IDPS). An example is wolfSSL's IDPS made in C (Pouzzner et al., 2023). Low-level languages provide more control over the system and network components, which is essential for developing IDPS. An IDPS is a network security tool that operates mainly at the network layer in the infrastructure. It is capable of understanding low-level traffic and protocols, making it effective in detecting and preventing attacks that may affect the web application, such as a DNS-server attack or a denial of service attack. (Thomassen, 2012)

Another area where low-level languages are used in cybersecurity is in the development of secure communication protocols, such as secure socket layer (SSL) and Transport Layer Security (TLS). According to Feldman (2018), different programming languages have various implementations of SSL and TLS, with OpenSSL being the most widely recognized. It is primarily written in C (TechTarget, 2014). These protocols provide secure communication channels between computers and networks by encrypting data that is transmitted over the network. Since these communication protocols are developed in a low-level language, it allows for faster execution compared to high-level languages.

Besides that, low-level languages can be used to develop malware (Nichols, 2020). For instance, malware writers may use low-level languages to create rootkits, which are stealthy programs that can hide their presence on a system and manipulate the behaviour of the operating system to avoid detection. Rootkits can be difficult to detect because they operate at the system level, making them difficult to distinguish from legitimate system components (Burdova, 2021). By using low-level programming languages like assembly language to develop malware, an expert programmer has the ability to write polymorphic code that alters itself while in operation, making it difficult to detect. The code can change its behaviour in unpredictable ways, exploit processor pipelining to act differently when under a debugger, and employ various other techniques that are not feasible in high-level programming languages. (Green, 2019)

Moreover, learning low-level programming languages such as C, C++, and assembly language is crucial for malware analysis in cybersecurity and forensics fields. Malware analysis involves studying malware to determine its function, origin, and potential impact, and is a critical aspect for companies that develop anti-virus or anti-malware solutions. C and C++ is commonly used in buffer overflow attacks and are one of the older programming languages that are widely used in creating malware. C has many windows-based libraries that efficiently control the computer's functionality, making it an important language to learn for effective malware analysis. Assembly language is a low-level programming language that humans can read, making it a valuable skill set for malware analysis. The lower down in the hierarchy of programming languages, the more details can be derived about how the malware works. Assembly language provides more details about how malware functions, making it important to learn how to read assembly code for effective malware analysis. (Brathwaite, 2021)

Finally, another area where low-level languages are used in cybersecurity is for reverse engineering. Although the term reverse engineering and malware analysis are often used interchangeably, they are not the same. Reverse engineering involves breaking down a system or product to understand its workings, while malware analysis focuses on studying malware to determine its behaviour and purpose. Thus, reverse engineering deals with understanding how a system works, while malware analysis focuses on understanding what the malware does. To reverse engineer malware, security researchers need to have expertise in low-level programming languages like assembly language. Assembly language can be used to write software that is close to the hardware which is mentioned previously. This proximity to the hardware makes assembly language well-suited for writing malware as it gives the attacker more control over the code's behavior. (Ec-Council, 2022) In order to do reverse engineering, a disassembler or decompiler is needed to analyze the malware code to understand its workings. One famous interactive disassembler used by many is IDA Pro by Hex Rays. IDA Pro is an advanced disassembler tool that helps in reverse engineering by creating visual maps of a program's execution and translating machine-executable code into a more human-readable form, assembly language. Its debugging feature has been improved with dynamic analysis capabilities, supporting multiple debugging targets, connection to both local and remote processes, and even support for 64-bit systems. However, since IDA Pro is quite pricey, Hex Rays do provide an alternative version of IDA called IDA Free but with limited features (Hex Rays, 2023).

In conclusion, low-level programming languages, such as C, C++, and assembly language, play a crucial role in the field of cybersecurity and forensics. They are commonly used in the development of IDPSs, secure communication protocols, malware analysis, and reverse engineering. These programming languages provide more control over the system and faster execution, making them important in identifying and addressing vulnerabilities in hardware and software. Furthermore, learning these programming languages is crucial for those in the cybersecurity and forensics fields as they provide more details about how malware functions, making it easier to study and determine its purpose.

3.0 SYSTEM DESIGN

3.1 - Main Section

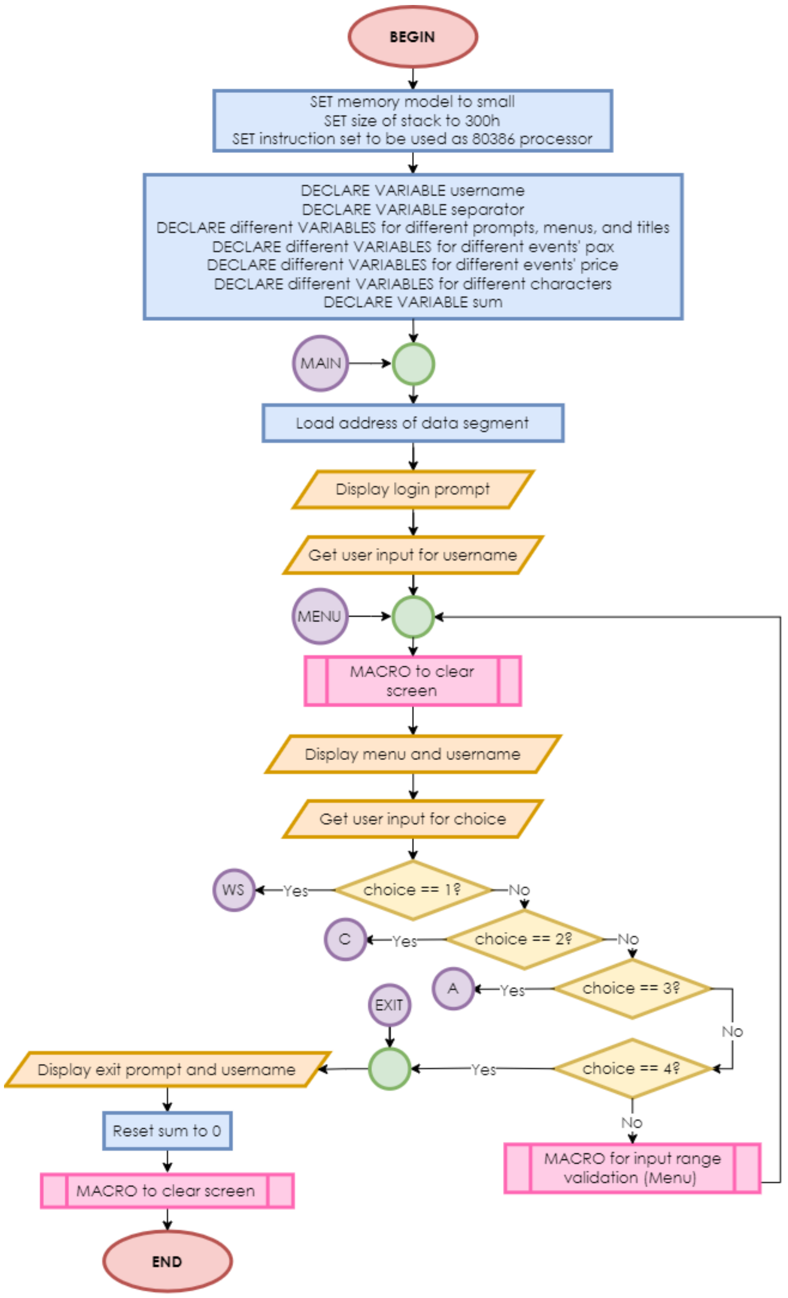

BEGIN
SET memory model to small
SET size of stack to 300h
SET instruction set to be used as 80386 processor
DECLARE VARIABLE username
DECLARE VARIABLE separator
DECLARE different VARIABLES for different prompts, menus, and titles
DECLARE different VARIABLES for different events' pax
DECLARE different VARIABLES for different events' price
DECLARE different VARIABLES for different characters
DECLARE VARIABLE sum
MAIN
Load address of data segment
Display login prompt
Get user input for username
MENU
MACRO to clear screen
Display menu and username
Get user input for choice
choice == 1?
WS
Yes
No
choice == 2?
C
choice == 3?
A
EXIT
choice == 4?
Display exit prompt and username
Reset sum to 0
MACRO to clear screen
END
MACRO for input range validation (Menu)

Figure 3.1 - Main Section Flowchart

The main section consists of the main flow without branches to the 4 main sections, namely the workshop, competition, activity, and checkout sections. The flow begins by setting the necessary memory model, size of stack, and instruction set (i386 processor) to be used. Then, it declares the necessary variables for the whole program in the data section. The address of the data section will then be loaded into the data segment register. A login prompt will then display for the user to input their name. Once, inputted, the main menu will show along with the inputted name, and it will prompt for a choice. Table 3.1 illustrates the outcome of the choices. The program will end when choice number 4 is chosen, or when anything that goes to the EXIT node from other sections.

Table 3.1 - Main Menu Choices

Choices	**Outcome**
1	Jump to Workshop Section (WS Node)
2	Jump to Competition Section (C Node)
3	Jump to Activity Section (A Node)
4	Exit program after resetting sum to 0
Otherwise	Jump to view main menu (MENU Node)

3.2 - Workshop Section

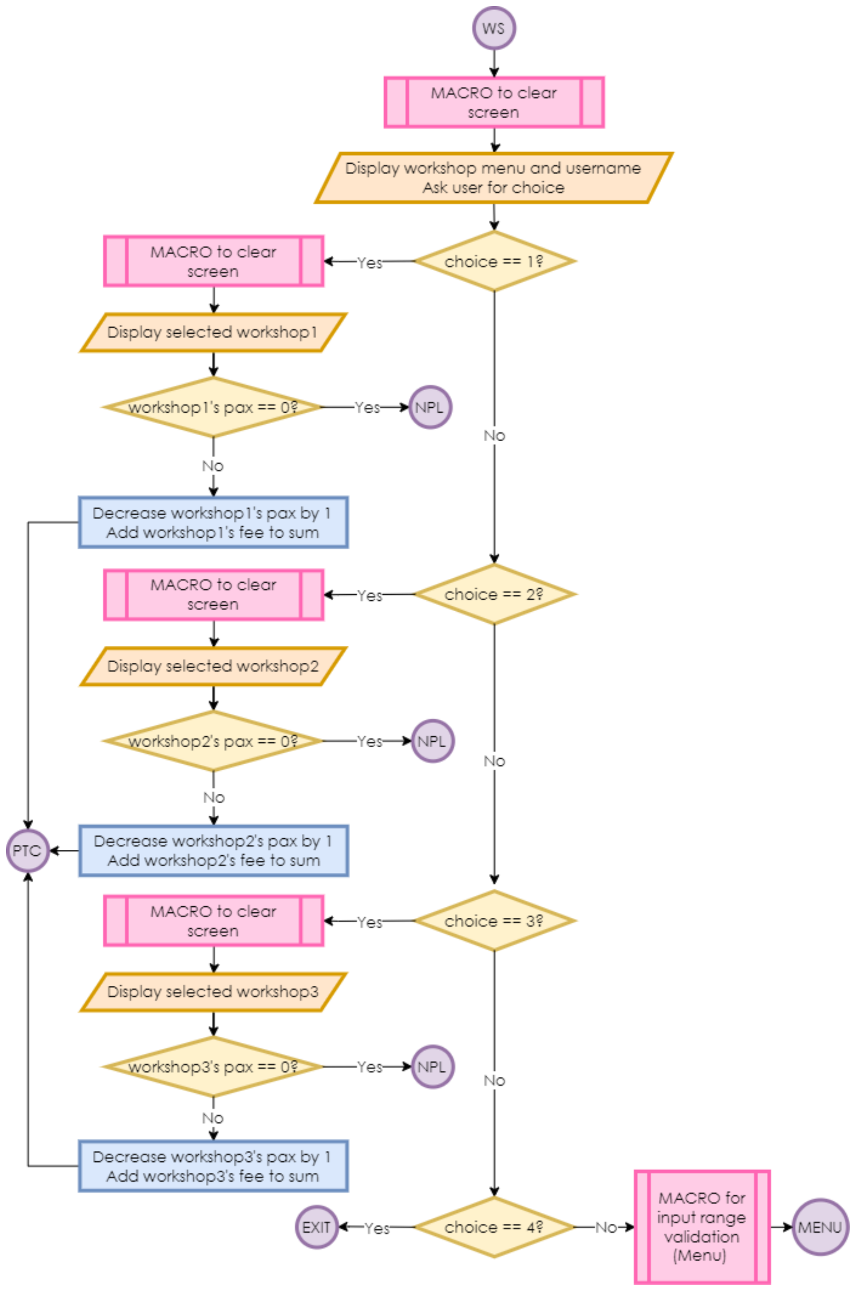

WS
MACRO to clear screen
Display workshop menu and username Ask user for choice
choice == 1?
Yes
No
MACRO to clear screen
Display selected workshop1
workshop1's pax == 0?
Yes
NPL
No
Decrease workshop1's pax by 1 Add workshop1's fee to sum
choice == 2?
Yes
No
MACRO to clear screen
Display selected workshop2
workshop2's pax == 0?
Yes
NPL
No
Decrease workshop2's pax by 1 Add workshop2's fee to sum
PTC
choice == 3?
Yes
No
MACRO to clear screen
Display selected workshop3
workshop3's pax == 0?
Yes
NPL
No
Decrease workshop3's pax by 1 Add workshop3's fee to sum
choice == 4?
Yes
EXIT
No
MACRO for input range validation (Menu)
MENU

Figure 3.2 - Workshop Section Flowchart

This section is the continuation from the WS node from the Main Section. The flow begins by showing the workshop menu which consists of the name, pax, price for the respective workshops as well as the username of the user. The program will then prompt for a choice. Table 3.2 illustrates the outcome of the choices. Choosing number 1 to 3 will display the selected workshop prompt. If the current workshop's pax is not zero, it will decrease the workshop's pax, and add the respective workshop fees to the sum variable. It will then proceed to jump to the PTC node for checkout prompt. If the current workshop's pax is zero, it will jump to the NPL node. The program will end when choice number 4 is chosen.

Table 3.2 - Workshop Menu Choices

Choices	**Outcome**
1	Opt for Workshop 1 (Windows x86 Buffer Overflow Workshop)
2	Opt for Workshop 2 (Pentesting 101 Workshop)
3	Opt for Workshop 3 (Exploit Development 101 Python Workshop)
4	Exit program after resetting sum to 0 (EXIT Node)
Otherwise	Jump to view main menu (MENU Node)

3.3 - Competition Section

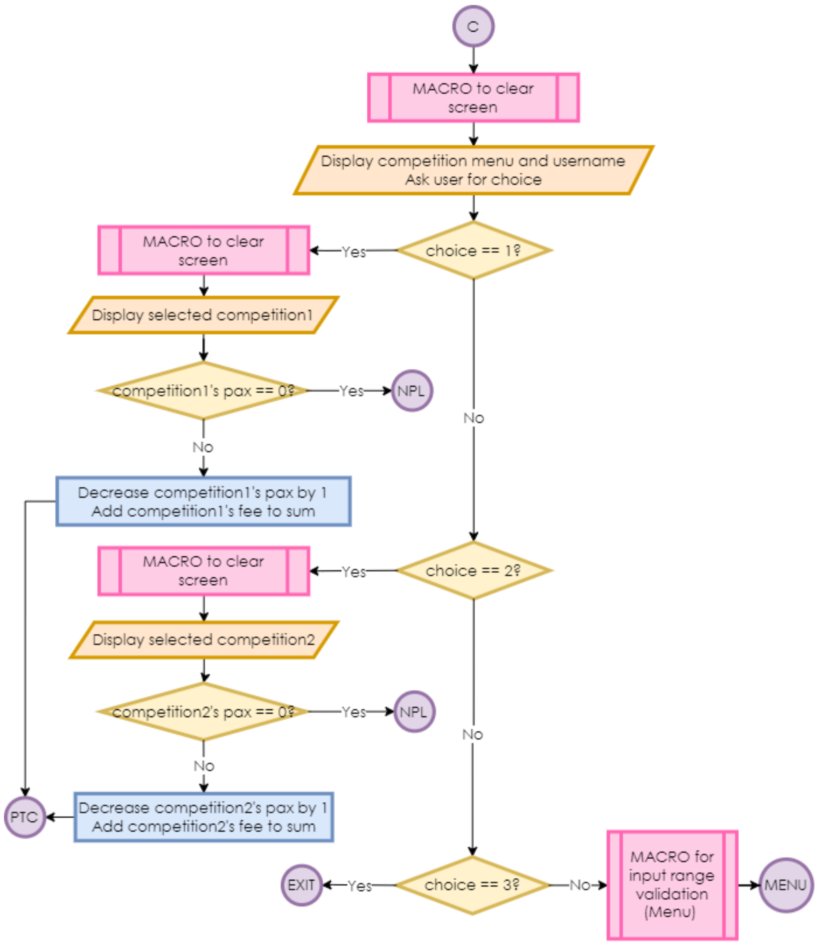

C
MACRO to clear screen
Display competition menu and username
Ask user for choice
choice == 1?
Yes
MACRO to clear screen
Display selected competition1
competition1's pax == 0?
Yes
NPL
No
Decrease competition1's pax by 1
Add competition1's fee to sum
No
choice == 2?
Yes
MACRO to clear screen
Display selected competition2
competition2's pax == 0?
Yes
NPL
No
Decrease competition2's pax by 1
Add competition2's fee to sum
PTC
No
choice == 3?
Yes
EXIT
No
MACRO for input range validation (Menu)
MENU

Figure 3.3 - Competition Section Flowchart

This section is the continuation from the C node from the Main Section. The flow begins by showing the competition menu which consists of the name, pax, price for the respective competitions as well as the username of the user. The program will then prompt for a choice. Table 3.3 illustrates the outcome of the choices. Choosing number 1 to 2 will display the selected competition prompt. If the current competition's pax is not zero, it will decrease the competition's pax, and add the respective competition fees to the sum variable. It will then proceed to jump to the PTC node for checkout prompt. If the current competition's pax is zero, it will jump to the NPL node. The program will end when choice number 4 is chosen.

Table 3.3 - Competition Menu Choices

Choices	Outcome

1	Opt for Competition 1 (Internal CTF 2023)
2	Opt for Competition 2 (Battle of Hackers CTF 2023)
4	Exit program after resetting sum to 0 (EXIT Node)
Otherwise	Jump to view main menu (MENU Node)

3.4 - Activity Section

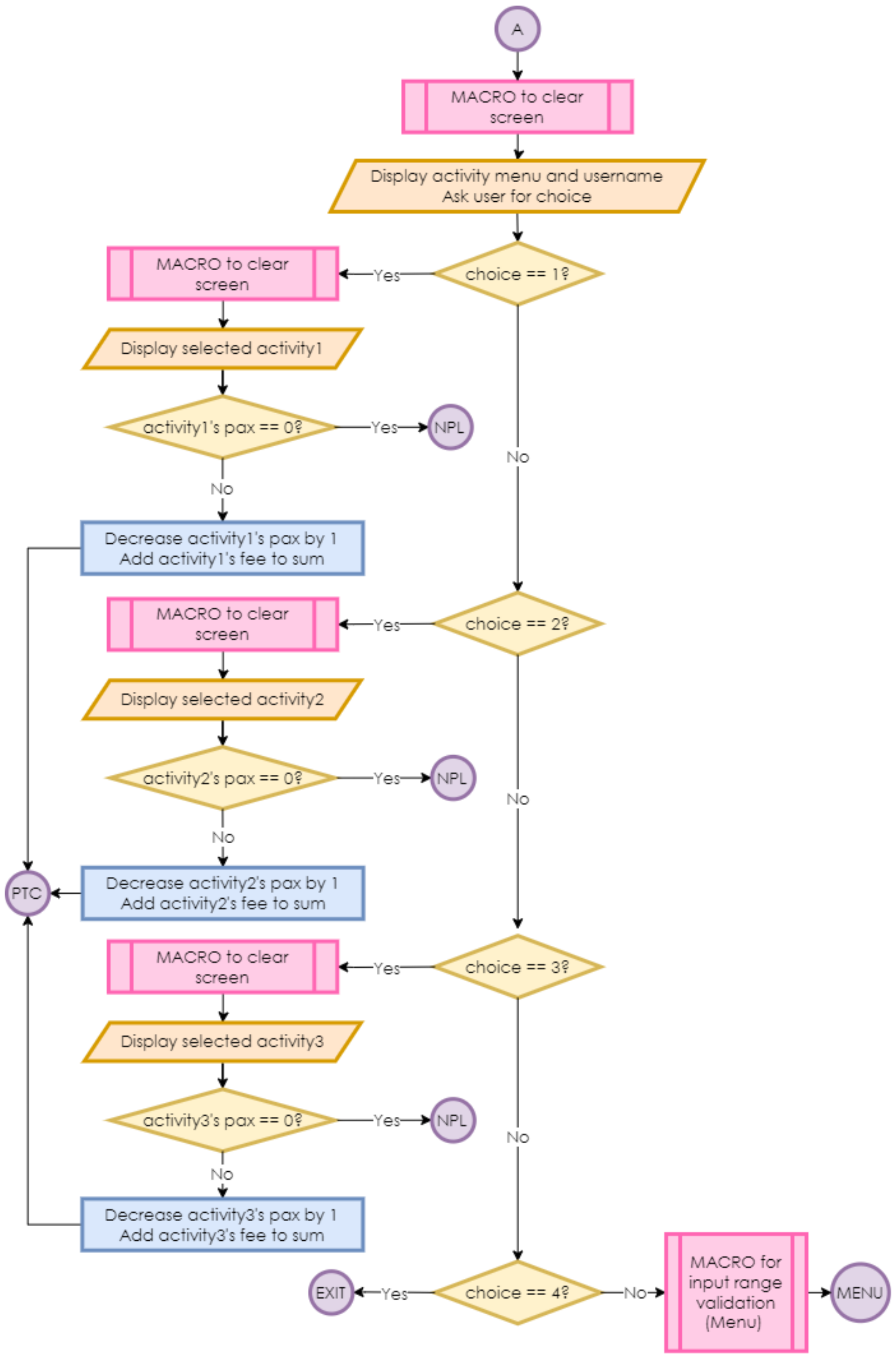

A
MACRO to clear screen
Display activity menu and username
Ask user for choice
choice == 1?
Yes
No
MACRO to clear screen
Display selected activity1
activity1's pax == 0?
Yes
NPL
No
Decrease activity1's pax by 1
Add activity1's fee to sum
choice == 2?
Yes
No
MACRO to clear screen
Display selected activity2
activity2's pax == 0?
Yes
NPL
No
Decrease activity2's pax by 1
Add activity2's fee to sum
PTC
choice == 3?
Yes
No
MACRO to clear screen
Display selected activity3
activity3's pax == 0?
Yes
NPL
No
Decrease activity3's pax by 1
Add activity3's fee to sum
EXIT
Yes
choice == 4?
No
MACRO for input range validation (Menu)
MENU

Figure 3.4 - Activity Section Flowchart

This section is the continuation from the A node from the Main Section. The flow begins by showing the activity menu which consists of the name, pax, price for the respective activities as well as the username of the user. The program will then prompt for a choice. Table 3.4 illustrates the outcome of the choices.

Choosing number 1 to 3 will display the selected activity prompt. If the current activity's pax is not zero, it will decrease the activity's pax, and add the respective activity fees to the sum variable. It will then proceed to jump to the PTC node for checkout prompt. If the current activity's pax is zero, it will jump to the NPL node. The program will end when choice number 4 is chosen.

Table 3.4 - Activity Menu Choices

Choices	**Outcome**
1	Opt for Activity 1 (REBoot Camp)
2	Opt for Activity 2 (CTF Crash Course)
3	Opt for Activity 3 (Across Verticals Company Visit)
4	Exit program after resetting sum to 0 (EXIT Node)
Otherwise	Jump to view main menu (MENU Node)

3.5 - Checkout Section

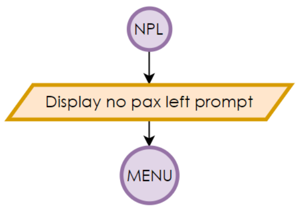


Figure 3.5.1 - No Pax Left (NPL) Flowchart

This section is the continuation from the NPL node from the Workshop, Competition or Activity Section. The flow begins by displaying that there are no pax left, and proceed to the main menu (MENU Node).

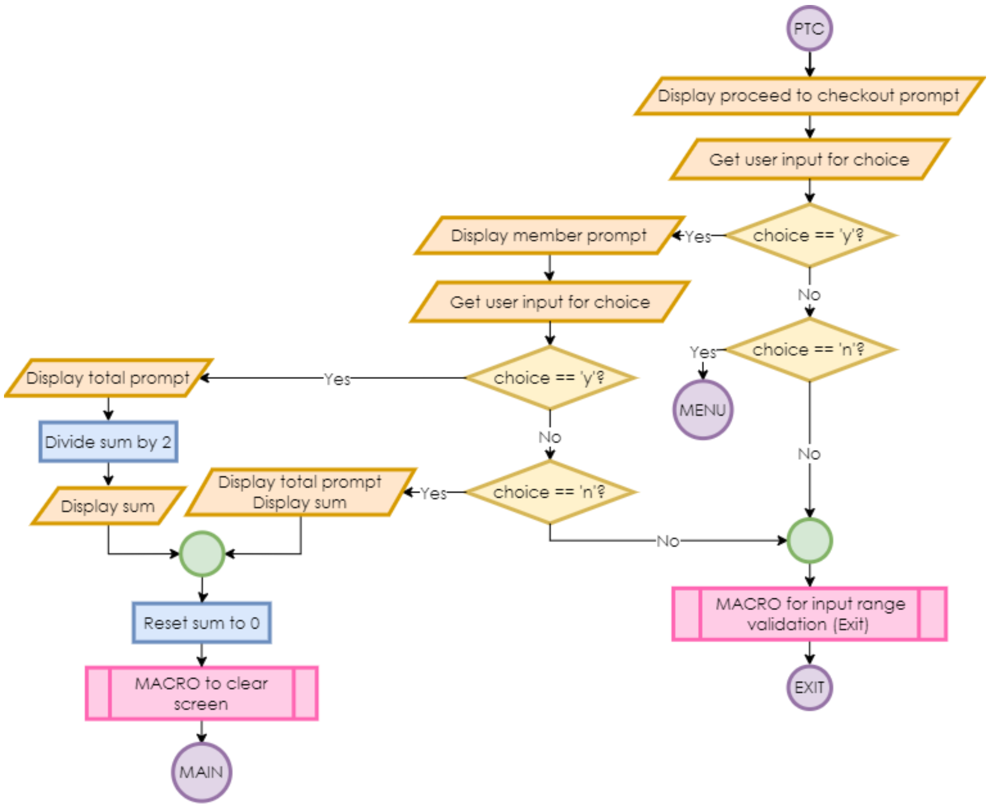


Figure 3.5.2 - Proceed To Checkout (PTC) Flowchart

This section is the continuation from the PTC node from the Workshop, Competition or Activity Section. The flow begins by displaying a prompt to checkout or not. The program will then prompt for a choice. Table 3.5.2.1 illustrates the outcome of the choices.

Table 3.5.2.1 - Proceed to Checkout Choices

Choices	Outcome
y	Proceed to checkout process
n	Jump to view main menu to register for other events (MENU Node)
Otherwise	Exit program after resetting sum to 0 (EXIT Node)

If the user chooses ‘y’ to checkout, the program will display a member prompt to check if the user is a FSEC-SS member or not. Table 3.5.2.2 illustrates the outcome of the choices. Both choices will display the total prompt and the calculated sum, then resets the sum to 0. It will then go to the MAIN node which directs to the Main Section, where another user can proceed to use the program.

Table 3.5.2.2 - FSEC-SS Member Prompt Choices

Choices	Outcome
y	Checkout with a 50% discount
n	Checkout with no discount
Otherwise	Exit program after resetting sum to 0 (EXIT Node)

3.6 - Miscellaneous Macros

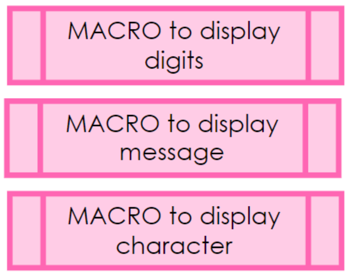


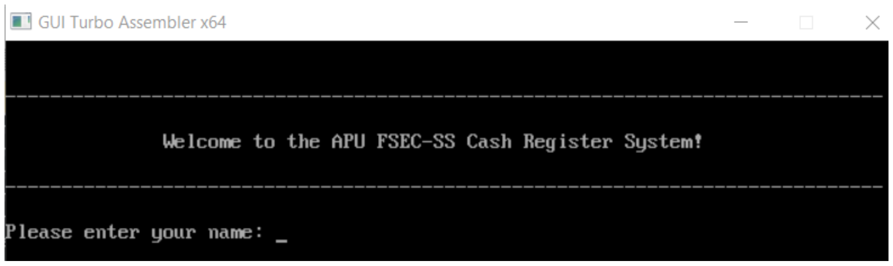
Figure 3.6 - Miscellaneous Macros

This section consists of macros that were not mentioned in the system design flowcharts since they are used throughout the program. They are display functions for digits, messages (string) and characters respectively.

4.0 SYSTEM SCREENSHOT

4.1 - Main Menu

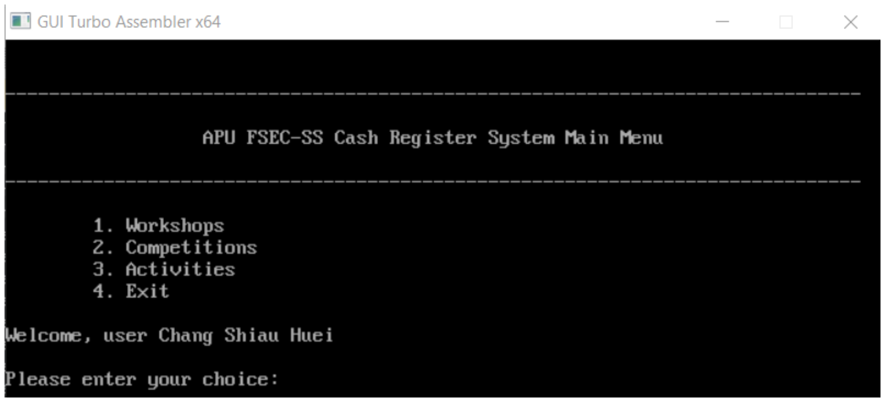
When entering the main menu, the user will be prompted to enter their name.



```
--------------------------------------------------------------------------------

              Welcome to the APU FSEC-SS Cash Register System!

--------------------------------------------------------------------------------

Please enter your name: _
```

Figure 4.1.1 - Login prompt

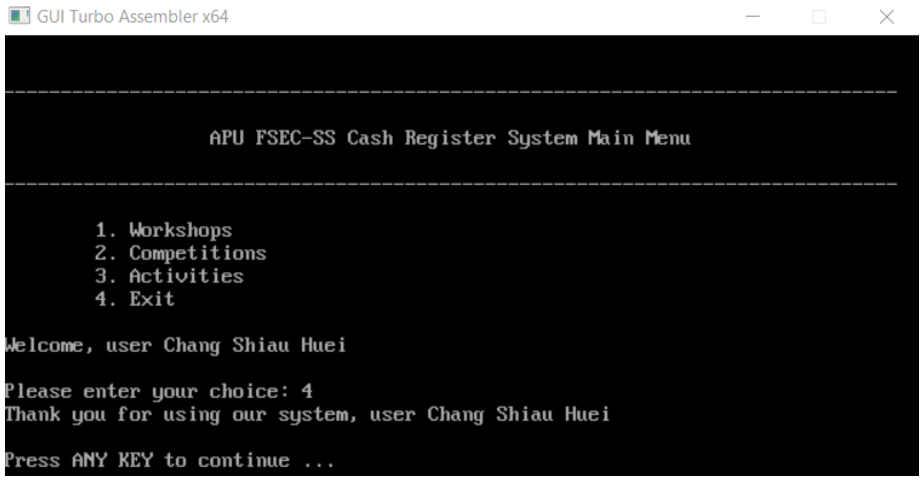
Once their name is entered, the main menu screen will be shown. The user can choose the events they would like to navigate to by entering their respective numbers. Inputting 1 will lead to the Workshop menu, 2 will lead to the Competition menu, and 3 will lead to the Activity menu.



```
------------------------------------------------------------------------------

                  APU FSEC-SS Cash Register System Main Menu

------------------------------------------------------------------------------

        1. Workshops
        2. Competitions
        3. Activities
        4. Exit

Welcome, user Chang Shiau Huei

Please enter your choice:
```

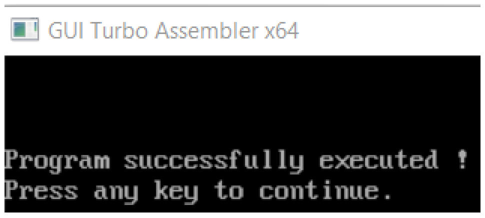
Figure 4.1.2 - Main menu

Inputting 4 will lead to the exit of the system, and the user could proceed to quit the program by inputting any key.



```
GUI Turbo Assembler x64
--------------------------------------------------------------------------------

                   APU FSEC-SS Cash Register System Main Menu

--------------------------------------------------------------------------------

        1. Workshops
        2. Competitions
        3. Activities
        4. Exit

Welcome, user Chang Shiau Huei

Please enter your choice: 4
Thank you for using our system, user Chang Shiau Huei

Press ANY KEY to continue ...
```

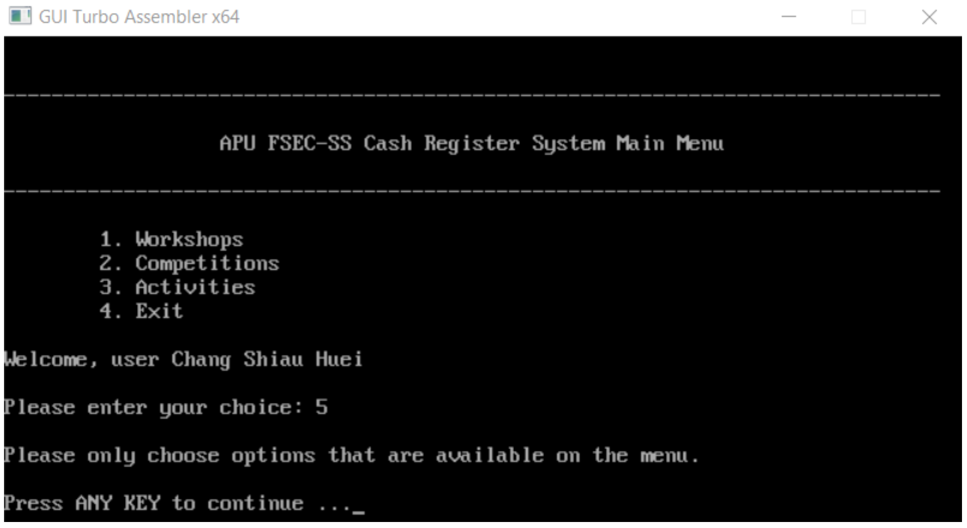
Figure 4.1.3 - Exit from Main menu



```
GUI Turbo Assembler x64

Program successfully executed !
Press any key to continue.
```

Figure 4.1.4 - Successfully exited program prompt

Inputting any other values than the ones mentioned will lead to an input validation prompt that tells the user to only choose options that are available on the menu. The user could proceed by inputting any key and they will be led to the main menu again.

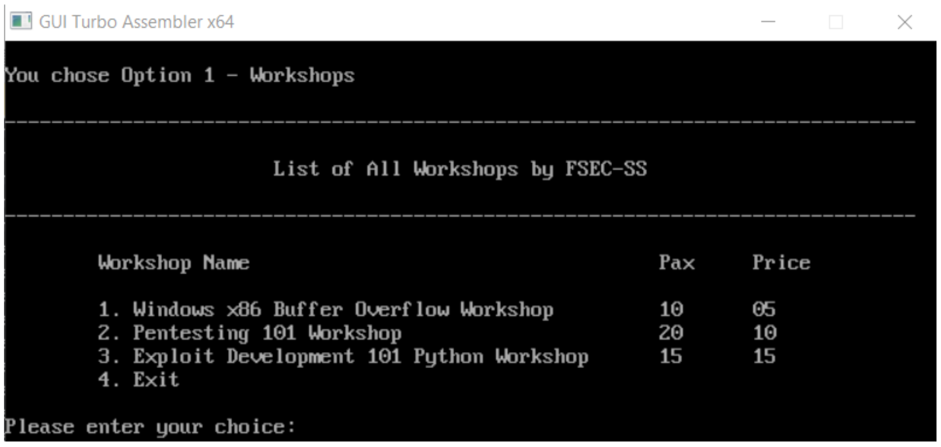


```
GUI Turbo Assembler x64
--------------------------------------------------------------------------------

                   APU FSEC-SS Cash Register System Main Menu

--------------------------------------------------------------------------------

        1. Workshops
        2. Competitions
        3. Activities
        4. Exit

Welcome, user Chang Shiau Huei

Please enter your choice: 5

Please only choose options that are available on the menu.

Press ANY KEY to continue ..._
```

Figure 4.1.5 - Input validation prompt

4.2 - Workshop Menu

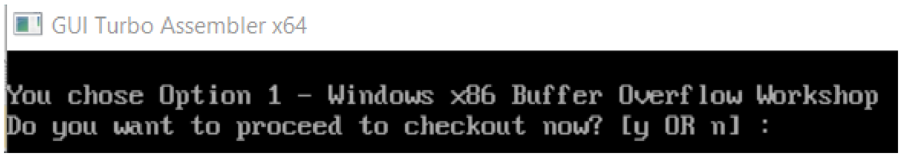
When entering the workshop menu, the user can choose the workshop they would like to register for by entering their respective numbers. The menu shows the workshop name, as well as their respective pax left and price.



```
You chose Option 1 - Workshops

------------------------------------------------------------------------------

                       List of All Workshops by FSEC-SS

------------------------------------------------------------------------------

        Workshop Name                                   Pax     Price

        1. Windows x86 Buffer Overflow Workshop         10      05
        2. Pentesting 101 Workshop                      20      10
        3. Exploit Development 101 Python Workshop      15      15
        4. Exit

Please enter your choice:
```

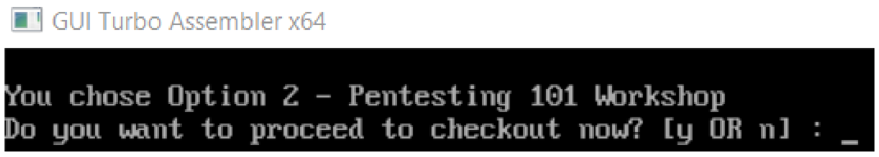
Figure 4.2.1 - Workshop menu

Inputting 1 will lead to Workshop 1 (Windows x86 Buffer Overflow Workshop), inputting 2 will lead to Workshop 2 (Pentesting 101 Workshop), and inputting 3 will lead to Workshop 3 (Exploit Development 101 Python Workshop).



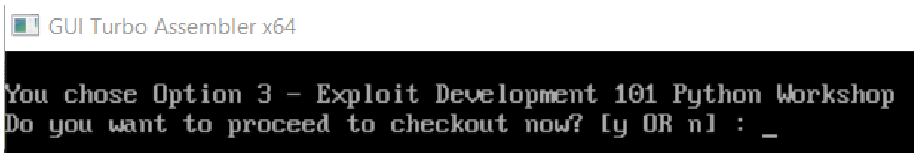
```
You chose Option 1 - Windows x86 Buffer Overflow Workshop
Do you want to proceed to checkout now? [y OR n] :
```

Figure 4.2.2 - Workshop 1 prompt



```
You chose Option 2 - Pentesting 101 Workshop
Do you want to proceed to checkout now? [y OR n] : _
```

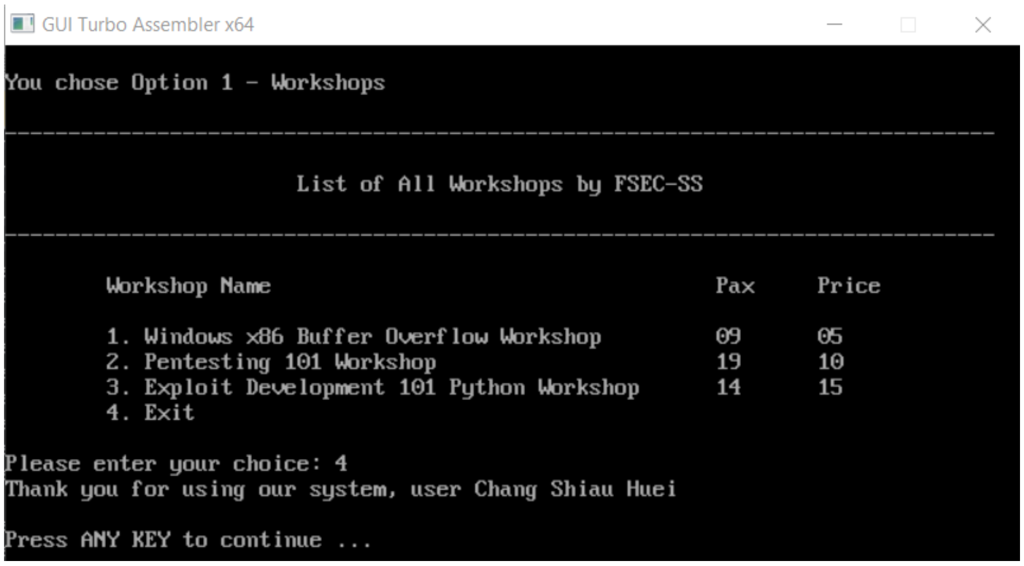
Figure 4.2.3 - Workshop 2 prompt



```
You chose Option 3 - Exploit Development 101 Python Workshop
Do you want to proceed to checkout now? [y OR n] : _
```

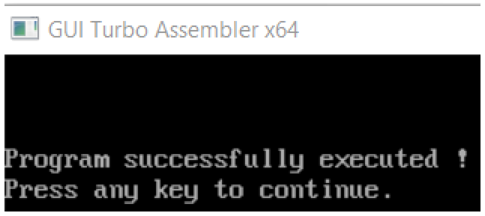
Figure 4.2.4 - Workshop 3 prompt

Inputting 4 will lead to the exit of the system, and the user could proceed to quit the program by inputting any key.



```
You chose Option 1 - Workshops
--------------------------------------------------------------------------------
                       List of All Workshops by FSEC-SS
--------------------------------------------------------------------------------
        Workshop Name                                   Pax     Price

        1. Windows x86 Buffer Overflow Workshop         09      05
        2. Pentesting 101 Workshop                      19      10
        3. Exploit Development 101 Python Workshop      14      15
        4. Exit

Please enter your choice: 4
Thank you for using our system, user Chang Shiau Huei

Press ANY KEY to continue ...
```

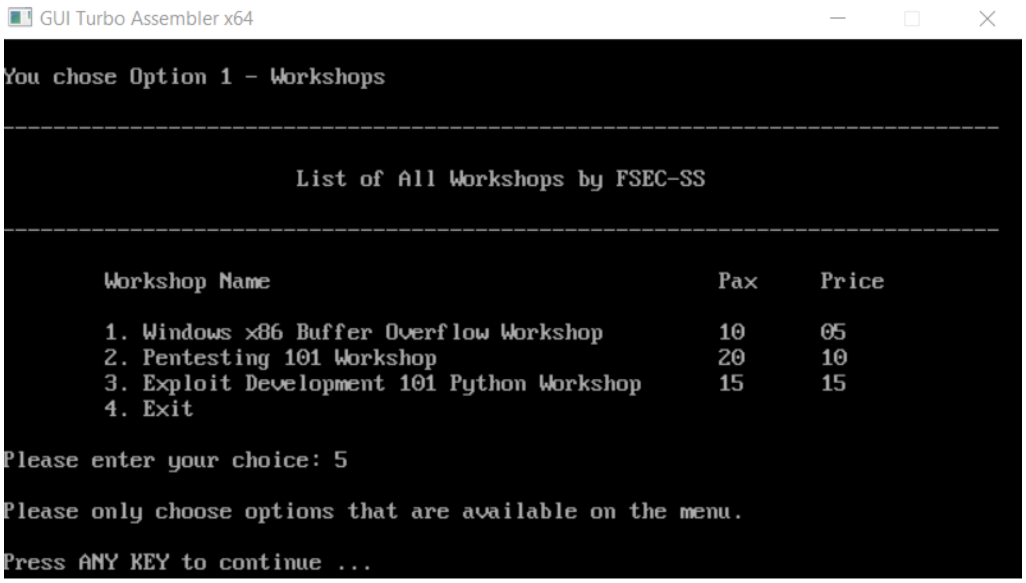
Figure 4.2.5 - Exit from Workshop menu



```
Program successfully executed !
Press any key to continue.
```

Figure 4.2.6 - Successfully exited program prompt

Inputting any other values than the ones mentioned will lead to an input validation prompt that tells the user to only choose options that are available on the menu. The user could proceed by inputting any key and they will be led to the main menu again.



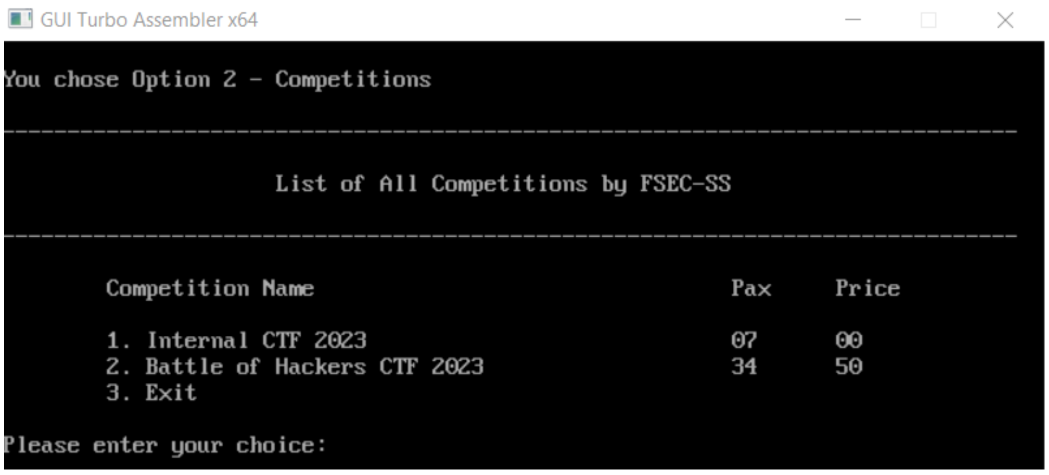
GUI Turbo Assembler x64

```
You chose Option 1 - Workshops

------------------------------------------------------------------------------

                       List of All Workshops by FSEC-SS

------------------------------------------------------------------------------

        Workshop Name                                   Pax      Price

        1. Windows x86 Buffer Overflow Workshop         10       05
        2. Pentesting 101 Workshop                      20       10
        3. Exploit Development 101 Python Workshop      15       15
        4. Exit

Please enter your choice: 5

Please only choose options that are available on the menu.

Press ANY KEY to continue ...
```

Figure 4.2.7 - Input validation prompt

4.3 - Competition Menu

When entering the competition menu, the user can choose the workshop they would like to register for by entering their respective numbers. The menu shows the competition name, as well as their respective pax left and price.

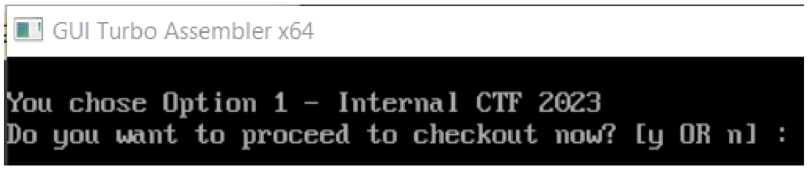


GUI Turbo Assembler x64

```
You chose Option 2 - Competitions

------------------------------------------------------------------------------

                    List of All Competitions by FSEC-SS

------------------------------------------------------------------------------

       Competition Name                              Pax     Price

       1. Internal CTF 2023                          07      00
       2. Battle of Hackers CTF 2023                 34      50
       3. Exit

Please enter your choice:
```

Figure 4.3.1 - Competition menu

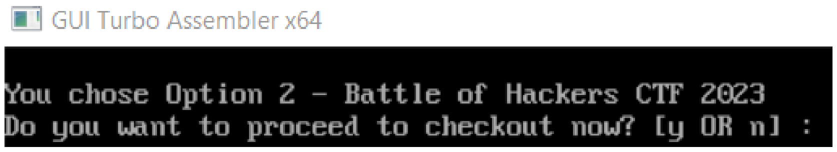
Inputting 1 will lead to Competition 1 (Internal CTF 2023), and inputting 2 will lead to Competition 2 (Battle of Hackers CTF 2023).



GUI Turbo Assembler x64

```
You chose Option 1 - Internal CTF 2023
Do you want to proceed to checkout now? [y OR n] :
```

Figure 4.3.2 - Competition 1 prompt

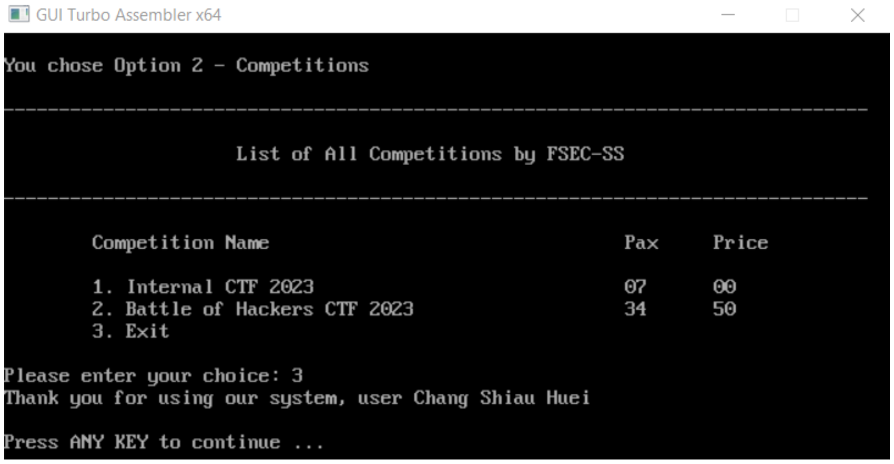


GUI Turbo Assembler x64

```
You chose Option 2 - Battle of Hackers CTF 2023
Do you want to proceed to checkout now? [y OR n] :
```

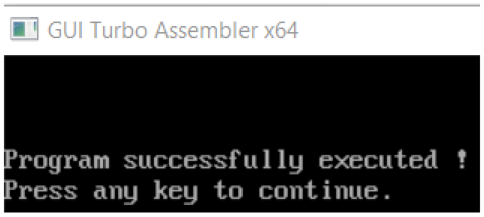
Figure 4.3.3 - Competition 2 prompt

Inputting 4 will lead to the exit of the system, and the user could proceed to quit the program by inputting any key.



```
GUI Turbo Assembler x64

You chose Option 2 - Competitions

--------------------------------------------------------------------------------

                    List of All Competitions by FSEC-SS

--------------------------------------------------------------------------------

        Competition Name                                Pax     Price

        1. Internal CTF 2023                            07      00
        2. Battle of Hackers CTF 2023                   34      50
        3. Exit

Please enter your choice: 3
Thank you for using our system, user Chang Shiau Huei

Press ANY KEY to continue ...
```

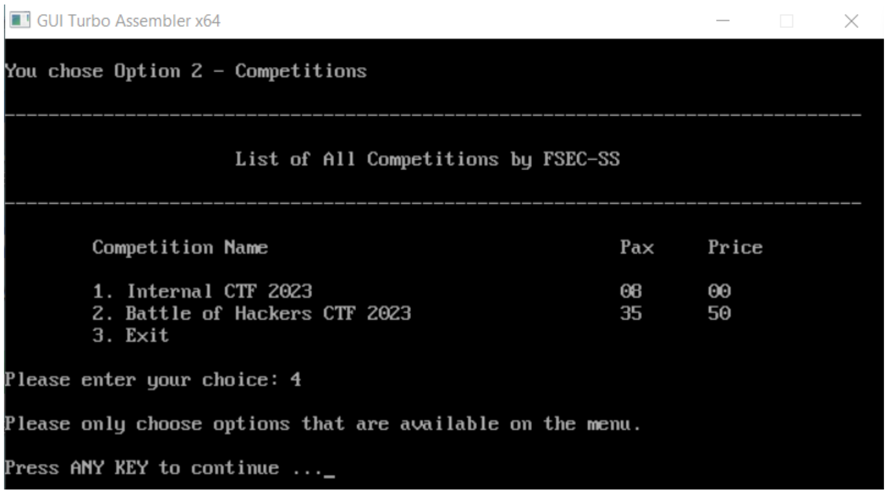
Figure 4.3.4 - Exit from Competition menu



```
GUI Turbo Assembler x64

Program successfully executed !
Press any key to continue.
```

Figure 4.3.5 - Successfully exited program prompt

Inputting any other values than the ones mentioned will lead to an input validation prompt that tells the user to only choose options that are available on the menu. The user could proceed by inputting any key and they will be led to the main menu again.

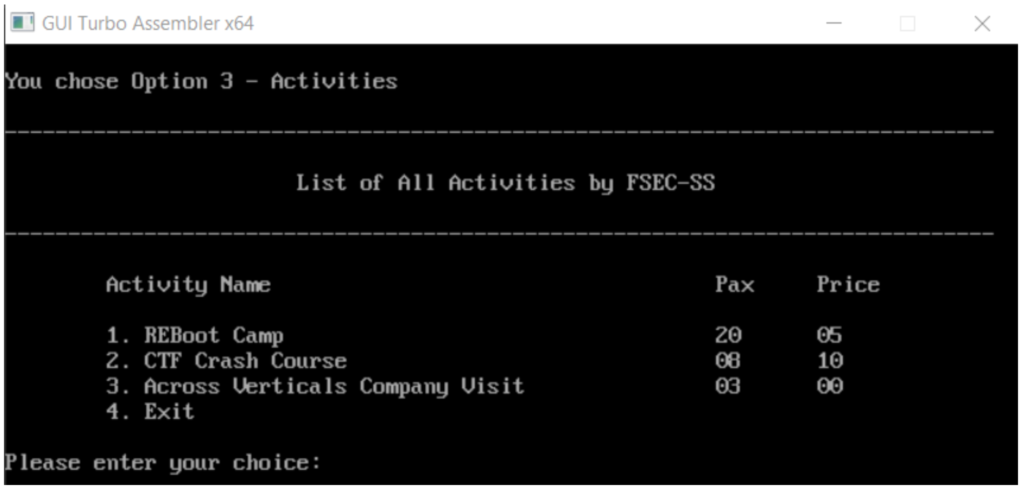


```
GUI Turbo Assembler x64

You chose Option 2 - Competitions

--------------------------------------------------------------------------------

                    List of All Competitions by FSEC-SS

--------------------------------------------------------------------------------

        Competition Name                                Pax     Price

        1. Internal CTF 2023                            08      00
        2. Battle of Hackers CTF 2023                   35      50
        3. Exit

Please enter your choice: 4

Please only choose options that are available on the menu.

Press ANY KEY to continue ..._
```

Figure 4.3.6 - Input validation prompt

4.4 - Activity Menu

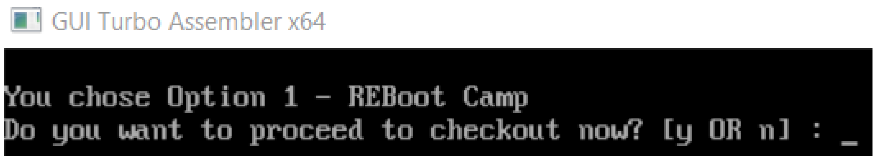
When entering the activity menu, the user can choose the activity they would like to register for by entering their respective numbers. The menu shows the activity name, as well as their respective pax left and price.



```
You chose Option 3 - Activities

---------------------------------------------------------------------------------

                       List of All Activities by FSEC-SS

---------------------------------------------------------------------------------

        Activity Name                                   Pax      Price

        1. REBoot Camp                                  20       05
        2. CTF Crash Course                             08       10
        3. Across Verticals Company Visit               03       00
        4. Exit

Please enter your choice:
```

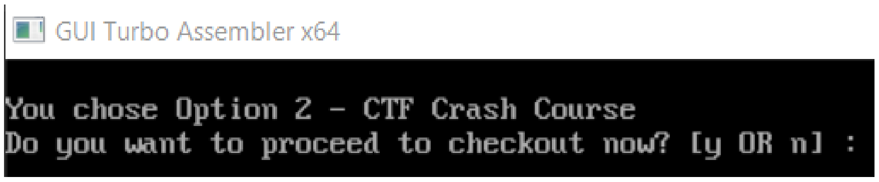
Figure 4.4.1 - Activity menu

Inputting 1 will lead to Activity 1 (REBoot Camp), inputting 2 will lead to Activity 2 (CTF Crash Course), and inputting 3 will lead to Activity 3 (Across Verticals Company Visit).



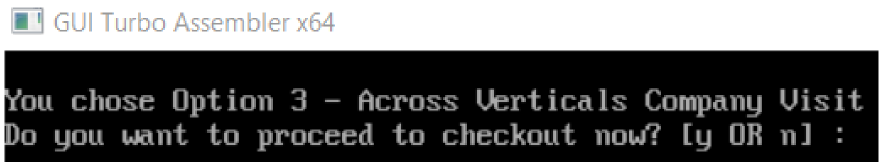
```
You chose Option 1 - REBoot Camp
Do you want to proceed to checkout now? [y OR n] : _
```

Figure 4.4.2 - Activity 1 prompt



```
You chose Option 2 - CTF Crash Course
Do you want to proceed to checkout now? [y OR n] :
```

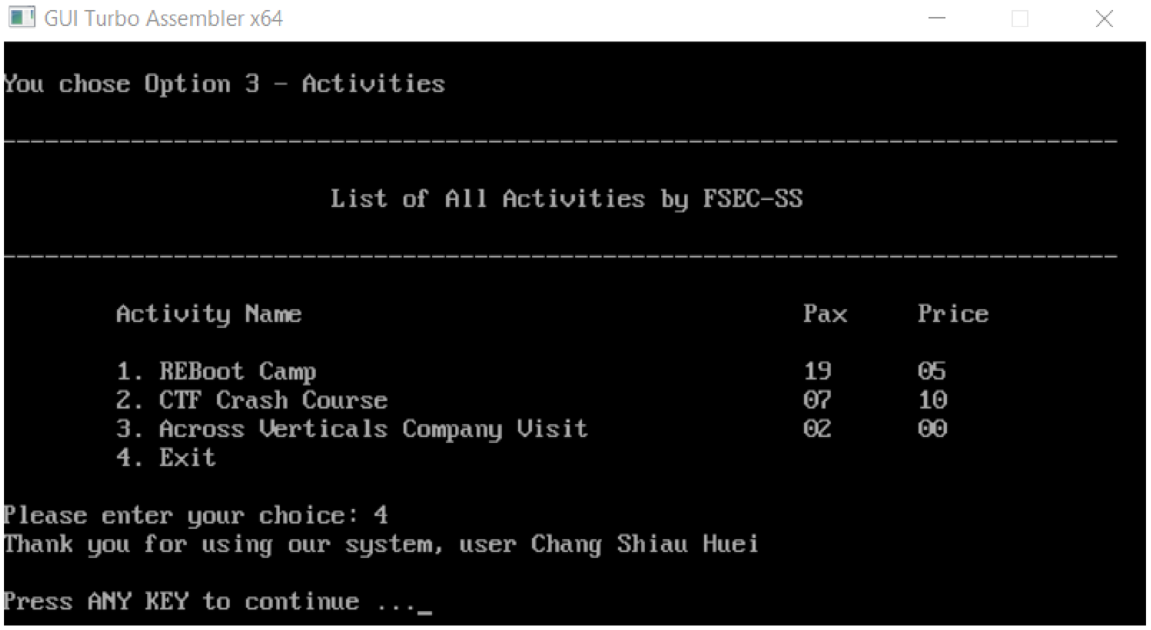
Figure 4.4.3 - Activity 2 prompt



```
You chose Option 3 - Across Verticals Company Visit
Do you want to proceed to checkout now? [y OR n] :
```

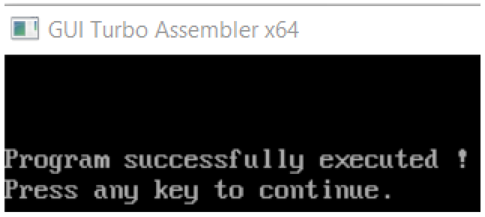
Figure 4.4.4 - Activity 3 prompt

Inputting 4 will lead to the exit of the system, and the user could proceed to quit the program by inputting any key.


GUI Turbo Assembler x64

```
You chose Option 3 - Activities

------------------------------------------------------------------------------

                       List of All Activities by FSEC-SS

------------------------------------------------------------------------------

        Activity Name                                   Pax     Price

        1. REBoot Camp                                  19      05
        2. CTF Crash Course                             07      10
        3. Across Verticals Company Visit               02      00
        4. Exit

Please enter your choice: 4
Thank you for using our system, user Chang Shiau Huei

Press ANY KEY to continue ..._
```

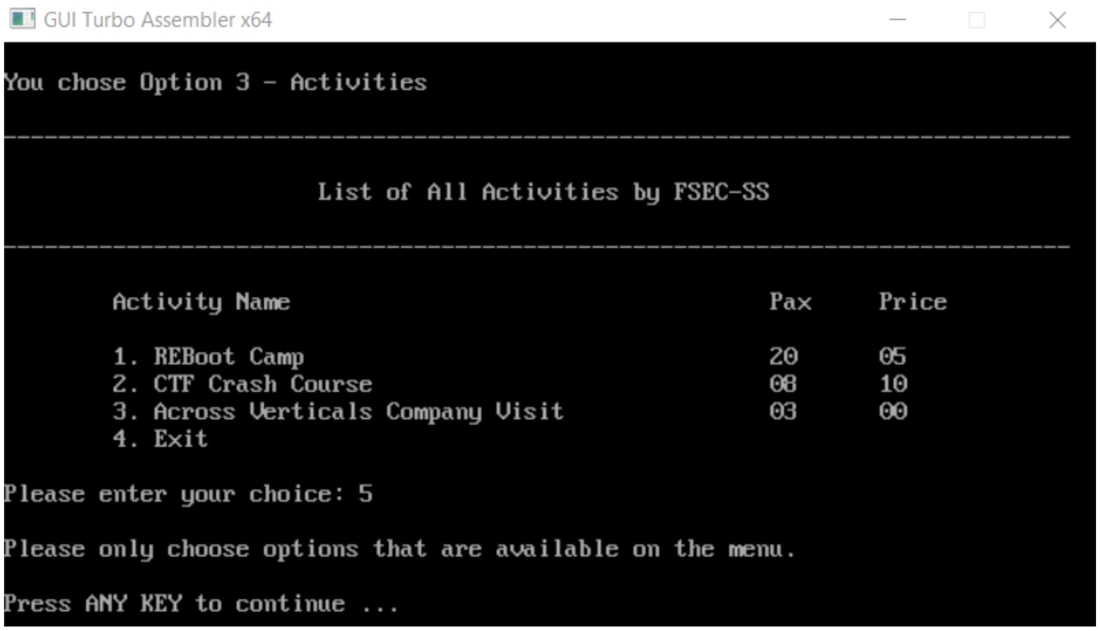
Figure 4.2.5 - Exit from Activity menu


GUI Turbo Assembler x64

```
Program successfully executed !
Press any key to continue.
```

Figure 4.2.6 - Successfully exited program prompt

Inputting any other values than the ones mentioned will lead to an input validation prompt that tells the user to only choose options that are available on the menu. The user could proceed by inputting any key and they will be led to the main menu again.

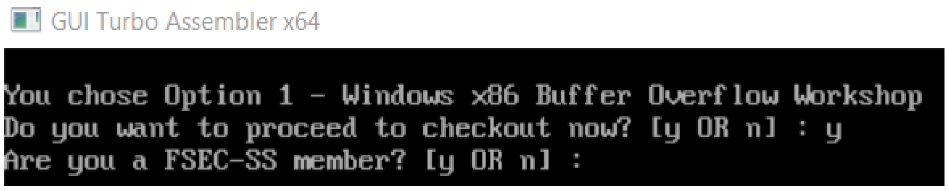


```
You chose Option 3 - Activities

------------------------------------------------------------------------------

                       List of All Activities by FSEC-SS

------------------------------------------------------------------------------

        Activity Name                                   Pax     Price

        1. REBoot Camp                                  20      05
        2. CTF Crash Course                             08      10
        3. Across Verticals Company Visit               03      00
        4. Exit

Please enter your choice: 5

Please only choose options that are available on the menu.

Press ANY KEY to continue ...
```

Figure 4.2.7 - Input validation prompt

4.5 - Checkout

After choosing a workshop, competition, or activity, the user will be prompted to checkout. If the user inputs ‘n’ (no), the user will be brought back the to Main menu. If the user inputs ‘y’ (yes), the user will be asked whether they are a FSEC-SS member or not.

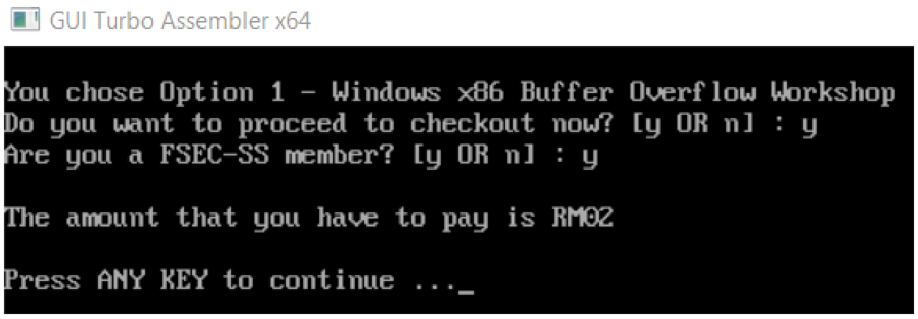

GUI Turbo Assembler x64

```
You chose Option 1 - Windows x86 Buffer Overflow Workshop
Do you want to proceed to checkout now? [y OR n] : y
Are you a FSEC-SS member? [y OR n] :
```

Figure 4.5.1 - Proceed to checkout and FSEC-SS member prompt

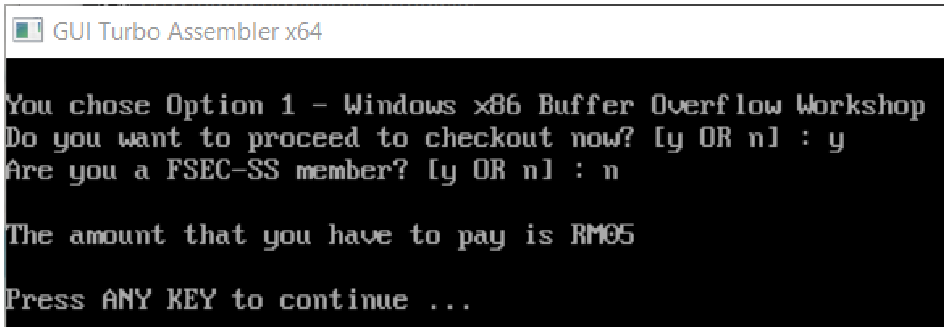
If the user is a FSEC-SS member and inputs ‘y’ (yes), the total amount that the user has to pay for the event fees is shown with a 50% discount. If the user is not a FSEC-SS member and inputs ‘n’ (no), the total amount that the user has to pay for the event fees is shown without any discount. In both situations, the user will then be able to go back to the login screen by inputting any key, so that another user could use the program.

NOTE: The program only shows the nearest ringgit and disregards the cents.


GUI Turbo Assembler x64

```
You chose Option 1 - Windows x86 Buffer Overflow Workshop
Do you want to proceed to checkout now? [y OR n] : y
Are you a FSEC-SS member? [y OR n] : y

The amount that you have to pay is RM02

Press ANY KEY to continue ..._
```

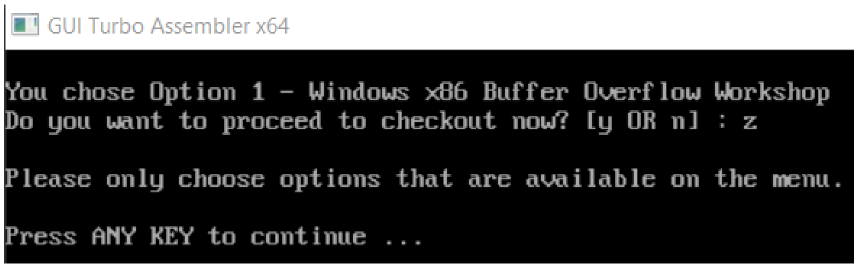
Figure 4.5.2 - 50% discount for FSEC-SS members


GUI Turbo Assembler x64

```
You chose Option 1 - Windows x86 Buffer Overflow Workshop
Do you want to proceed to checkout now? [y OR n] : y
Are you a FSEC-SS member? [y OR n] : n

The amount that you have to pay is RM05

Press ANY KEY to continue ...
```

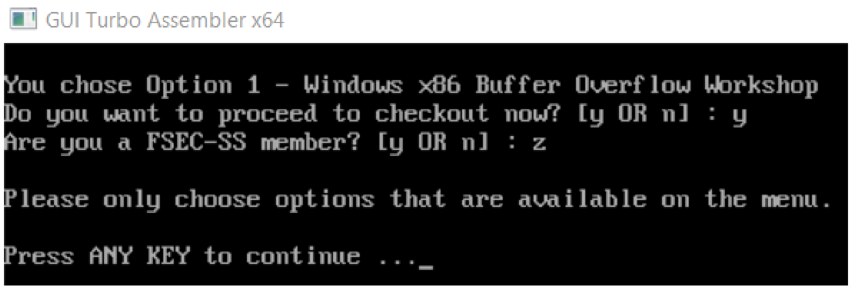
Figure 4.5.3 - No discount for non-FSEC-SS members

Inputting any other values than the ones mentioned will lead to an input validation prompt that tells the user to only choose options that are available on the menu. The user could proceed by inputting any key and they will exit the program.



GUI Turbo Assembler x64

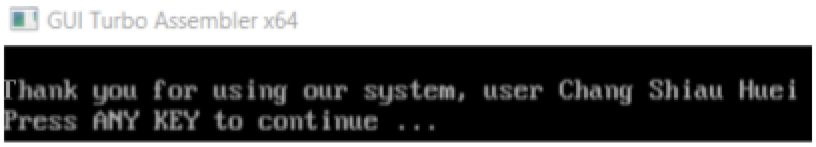
```
You chose Option 1 - Windows x86 Buffer Overflow Workshop
Do you want to proceed to checkout now? [y OR n] : z

Please only choose options that are available on the menu.

Press ANY KEY to continue ...
```



GUI Turbo Assembler x64

```
You chose Option 1 - Windows x86 Buffer Overflow Workshop
Do you want to proceed to checkout now? [y OR n] : y
Are you a FSEC-SS member? [y OR n] : z

Please only choose options that are available on the menu.

Press ANY KEY to continue ..._
```

Figure 4.5.4 & Figure 4.5.5 - Input validation prompt

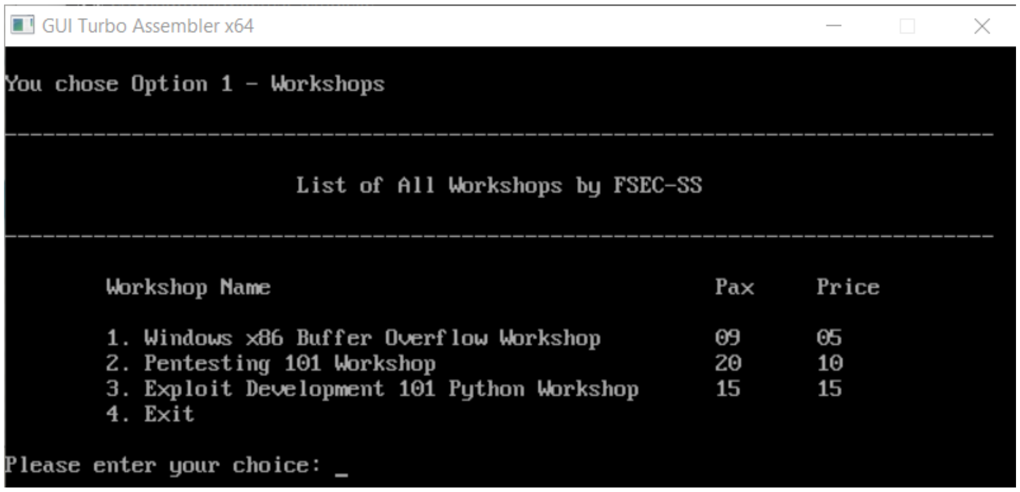


GUI Turbo Assembler x64

```
Thank you for using our system, user Chang Shiau Huei
Press ANY KEY to continue ...
```

Figure 4.5.6 - Exit program prompt

After selecting a workshop, competition, or activity, the pax of the respective event will decrement by 1 as shown in Figure 4.5.7.

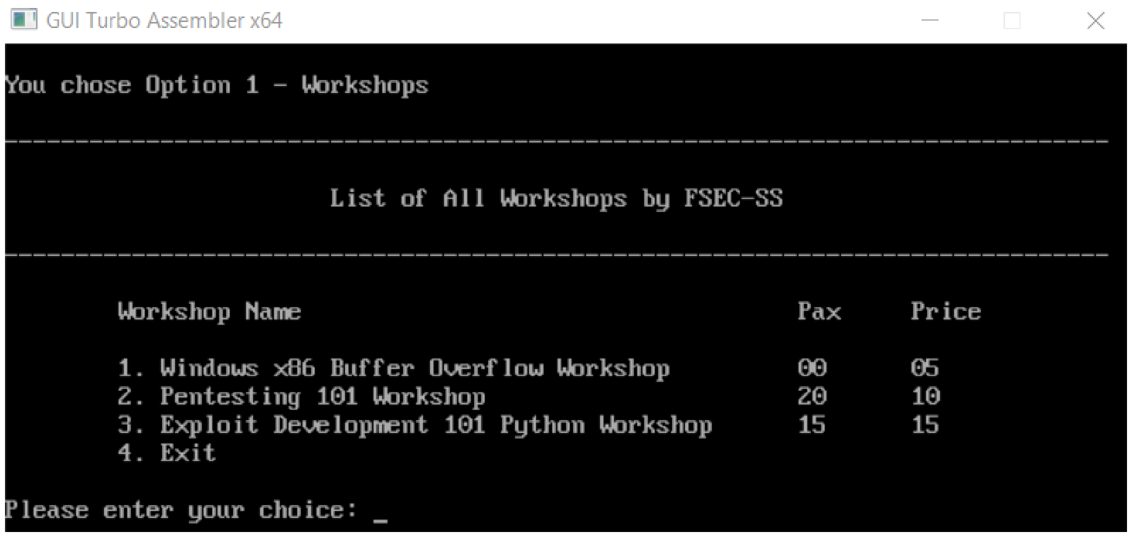


GUI Turbo Assembler x64

```
You chose Option 1 - Workshops

-----------------------------------------------------------------------------

                       List of All Workshops by FSEC-SS

-----------------------------------------------------------------------------

        Workshop Name                                Pax      Price

        1. Windows x86 Buffer Overflow Workshop      09       05
        2. Pentesting 101 Workshop                   20       10
        3. Exploit Development 101 Python Workshop   15       15
        4. Exit

Please enter your choice: _
```

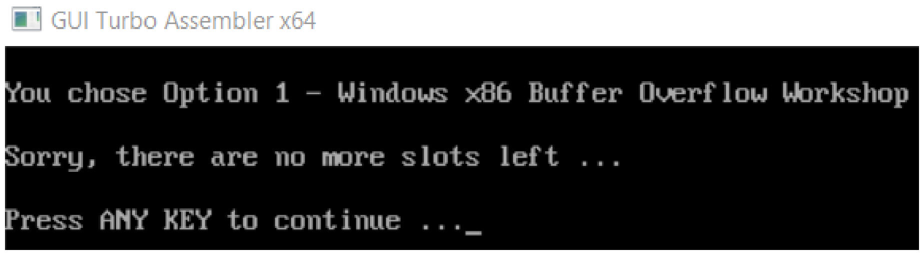
Figure 4.5.7 - Decrement of pax value

If the pax of the respective event is 0, a prompt that tells the user there are no more slots left will be shown. The user could proceed to the main menu by inputting any key.



```
You chose Option 1 - Workshops

------------------------------------------------------------------------------

                       List of All Workshops by FSEC-SS

------------------------------------------------------------------------------

        Workshop Name                                 Pax     Price

        1. Windows x86 Buffer Overflow Workshop       00      05
        2. Pentesting 101 Workshop                    20      10
        3. Exploit Development 101 Python Workshop    15      15
        4. Exit

Please enter your choice: _
```

Figure 4.5.8 - Pax value is 0



```
You chose Option 1 - Windows x86 Buffer Overflow Workshop

Sorry, there are no more slots left ...

Press ANY KEY to continue ..._
```

Figure 4.5.9 - No more slots prompt

5.0 CONCLUSION

In conclusion, low-level programming languages, including Assembly language, are becoming increasingly important, especially in the cybersecurity and forensic fields. Assembly language provides direct access to the computer hardware, allowing for low-level operations not available in higher-level programming languages, enabling professionals to fine-tune their code and achieve optimal performance in areas such as malware development and analysis, intrusion detection systems, firewalls, and reverse engineering, to name a few examples. As technology continues to advance and the need for robust cybersecurity and forensic solutions increases, the use of low-level programming languages, including Assembly language, is expected to grow.

6.0 SELF REFLECTION

Assembly language is a low-level programming language, which makes it well-suited for tasks that require direct access to the computer hardware. Developing a FSEC-SS cash register system using Assembly language has offered me the opportunity to demonstrate a strong understanding of the fundamentals of computer architecture and the inner workings of computer systems.

Assembly language can be challenging to work with, as it requires a high level of precision and requires the programmer to have a deep understanding of the underlying hardware. Coding a cash register system using Assembly language provides a valuable learning experience, as it requires the programmer to think about problem-solving in a different way than when using higher-level programming languages. The process of coding a cash register system in Assembly language requires patience and perseverance, as it is a complex and time-consuming task. However, the end result is a highly optimized and efficient solution that is tailored to the specific needs of the cash register system.

I would like to thank my lecturer and university for providing me with the opportunity to code a FSEC-SS Cash Register System using Assembly language as an assignment. This assignment has given me a deeper understanding of Assembly language and has allowed me to explore its applications in the field of computer science. The hands-on experience I gained while coding the cash register system has been invaluable, and I am grateful for the chance to expand my knowledge in this area. I am confident that the skills I have learned from this assignment will be useful in my future endeavours. Thank you again for your support and guidance.

7.0 REFERENCES

Asia Pacific University. (2023). *BSc (Hons) In Computer Science with a specialism in Digital Forensics | Asia Pacific University (APU)*. https://www.apu.edu.my/our-courses/undergraduate-studies/school-computing-technology/bsc-hons-computer-science-specialism-digital-forensics

Brathwaite, S. (2021). *Top Programming Languages For Malware Analysis*. Cybrary. https://www.cybrary.it/blog/top-programming-languages-for-malware-analysis/

Burdova, C. (2021). *What Is a Rootkit and How to Remove It?* Avast. https://www.avast.com/c-rootkit

Cempron, J. P., Salinas, C. S., & Uy, R. L. (2017). Assembly Program Performance Analysis Metrics: Instructions Performed and Program Latency Exemplified on Loop Unroll. *Philippine Journal of Science*, *147*(3), 441–452. https://philjournalsci.dost.gov.ph/images/pdf/pjs_pdf/vol147no3/assembly_program_optimization.pdf

Chen, J. (2021). *What Is High-Frequency Trading (HFT)? How It Works and Example*. Investopedia. https://www.investopedia.com/terms/h/high-frequency-trading.asp

Computer Hope. (2019). *What is a Low-Level Language?* https://www.computerhope.com/jargon/l/lowlangu.htm

Dodos, S. (2021). *Device Drivers | What, How, Types, Architecture & Summary*. Teach Computer Science. https://teachcomputerscience.com/device-drivers/

Ec-Council. (2022). *A Quick Guide to Reverse Engineering Malware*. Cybersecurity Exchange. https://www.eccouncil.org/cybersecurity-exchange/ethical-hacking/malware-reverse-engineering/

Eggleston, L. (2020). *A Guide to Low Level Programming for Beginners*. Course Report. https://www.coursereport.com/blog/a-guide-to-low-level-programming-for-beginners

Feldman, J. B. (2018). *What language is used in SSL?* Quora. https://www.quora.com/What-language-is-used-in-SSL

Fernando, J. (2022). *Assembly Language*. Investopedia. https://www.investopedia.com/terms/a/assembly-language.asp

Gonzalez, G. (2016). *Where is assembly language needed in operating system development? [Online forum post]*. Quora. https://www.quora.com/Where-is-assembly-language-needed-in-operating-system-development

Green, J. (2019). *Do you think that with Assembly, you can make malware that is more efficient, to the point, and harder to detect than, C/C++, or some other language more distant from the hardware? [Online forum post]*. Quora. https://www.quora.com/Do-you-think-that-with-Assembly-you-can-make-malware-that-is-more-efficient-to-the-point-and-harder-to-detect-than-C-C++-or-some-other-language-more-distant-from-the-hardware

Hex Rays. (2023). *Hex Rays - State-of-the-art binary code analysis solutions*. https://hex-rays.com/ida-pro/

Marchan, J. (2023). *How to "compile" a compiler*. Teldat. https://www.teldat.com/blog/compiler-key-element-in-developing-software/

Minaie, A., & Sanati-Mehrizy, R. (Eds.). (2003). *A New Role of Assembly Language in Computer Engineering/Science Curriculum*. American Society for Engineering Education.

Nichols, S. (2020). *So you've decided you want to write a Windows rootkit. Good thing this chap's just demystified it in a talk*. The Register. https://www.theregister.com/2020/08/07/def_con_demirkapi/

Pal, K. (2017). *Why is learning assembly language still important?* Techopedia.com. https://www.techopedia.com/why-is-learning-assembly-language-still-important/7/32268

Pedamkar, P. (2022). *What is Assembly Language?* EDUCBA. https://www.educba.com/what-is-assembly-language/

Pouzzner, D., Hutchings, A., Garske, D., & Miyazaki, H. (2023). *GitHub - wolfSSL/wolfsentry: wolfSSL Intrusion Detection and Prevention System (IDPS)*. GitHub. https://github.com/wolfSSL/wolfsentry

Preston, R. (2021). *What Is Assembly Language? (With Components and Example)*. Indeed. https://www.indeed.com/career-advice/career-development/what-is-assembly-language

TechTarget. (2014). *OpenSSL*. WhatIs.com. https://www.techtarget.com/whatis/definition/OpenSSL

Thomassen, P. (2012). *AppSensor*. Norwegian University of Science and Technology.