**Score: \_\_\_\_\_**

**LA1 – Translation and Interpretation**

**Activities**

COMP256 – Computing Abstractions

Dickinson College

Spring 2023

Prof. Grant Braught

**Name:**

Today’s class introduced the new topic of Language Abstractions. Each level in the language abstraction hierarchy defines a new abstract machine that can be programmed at its level of abstraction. We first saw this with machine language in the Machine Abstractions topic. Programming at the machine language level is an abstraction that allows us to focus more on the details that matter to us than when we were programming at the microinstruction level. For example, at the machine language level we can think about what opcode and operands we need while ignoring the microinstruction level details about things like which switches need to be opened or closed. We let the abstraction handle that for us!

We will now begin looking at machines at higher levels of abstraction (Assembly Language and High-level Language) and how programs written for these machines can be executed on a physical machine. *Translation* (compilation/assembly) and *interpretation* are the two primary mechanisms by which this happens. Though we will also see that a combination of the two is increasingly popular and has some of the advantages of each.

**Translation and Interpretation:**

There is variety in the perspectives and language that exist around the topics of program translation and interpretation. Thus, it is helpful to see and hear about these topics from a variety of sources. Each of the following videos provides some additional information and details about the processes, advantages and disadvantages of program translation and program interpretation. They are relatively short, so please watch all three. As you watch them identify the concepts that are similar across them and pay attention to the slightly different terminology that each uses to refer to these concepts.

* The video *Compiler and Interpreter: Compiled Language vs Interpreted Programming Languages* from Coding Mentors is a good introduction to the topic.
  + <https://www.youtube.com/watch?v=I1f45REi3k4> (6:04)
* The video *Compilation vs Interpretation* from BGZDevTips covers the same topics but with a slightly different take and some more illustrations.
  + <https://www.youtube.com/watch?v=JNMy969SjyU> (6:06)
* Jakob, on his blog Finematics provides some reading and a video *Compiled vs Interpreted Programming Languages*. This is probably the most comprehensive discussion and he writes everything out if you find that easier to follow. He also does a more complete job of explaining the virtual machine approach and Just In Time (JIT) compilation.
  + <https://finematics.com/compiled-vs-interpreted-programming-languages/> (9:33)

🔑 1. There are a number of key terms that are used throughout these videos. Draw on the content of these videos and from class to give a brief definition of each of the following terms using a few sentences of your own words.

a. Source Code

b. Executable (or binary) File

c. Intermediate Language (or Representation)

🔑 2. There are a number of different programs involved in translation and interpretation. Draw on the content of the videos and from class to describe what each of the following programs does using a few sentences of your own words.

a. Compiler

b. Assembler

c. Interpreter

d. Virtual Machine

🔑 3. The two primary approaches to the execution of high-level language programs, translation and interpretation, have their own advantages and disadvantages. Several of the videos above discuss these advantages and disadvantages. Draw on the information in those videos to place each advantage or disadvantage below into the appropriate cell in the table (e.g. place “Slower Program Execution” in one cell and “Faster Program Execution” in another cell.

* Slower Program Execution / Faster Program Execution
* Slower Development & Debugging / Faster Development & Debugging
* Source code is private / Source code is public
* Programs are Cross Platform (portable) / Programs are Platform Dependent (not portable)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  | **Advantages** | **Disadvantages** |  |
|  | **Translated** |  |  |  |
|  | **Interpreted** |  |  |  |
|  |  |  |  |  |

4. Consider each of the following scenarios in light of the advantages and disadvantages of interpreted and translated programs from question #3. Then for each scenario, indicate if a translated language or an interpreted language would be more appropriate. **Give a brief rationale for each of your choices**.

a. A small utility program that will be used by people with lots of different types of computers.

b. A program that the end users should be able to easily change to suit their specific situation.

c. A program, based on trade secrets, that you want people to run but not be able to modify.

d. A program to perform complex calculations on a very expensive high-performance super-computer.

e. A program that needs to be changed, updated and tested frequently.

**The Hybrid Approach:**

The intermediate or hybrid approach was pioneered by Java and the engineers at Sun Microsystems to try to capture some of the advantages of both translation and interpretation, while mitigating their disadvantages. It has since been applied to a wide variety of popular languages including C#, Python, Ruby, Kotlin and many others.

5. The hybrid approach combines both translation and interpretation with the translation taking place on the software developer’s machine and interpretation taking place on the end user’s machine.

a. Write a sentence using the terms **source code, compiler and intermediate language** that describes what happens *on the developer’s machine* during the translation stage in the hybrid approach.

b. Write a sentence using the terms **intermediate language (or bytecode), virtual machine, interpreter and machine language** that describes what happens *on the end user’s machine* during the interpretation stage in the hybrid approach**.**

🏆 6. In a few sentences of your own words, explain how the hybrid approach captures some of the benefits of both interpreted programs and translated programs.

7. For each of the following, indicate if it is most likely to be source code, bytecode or executable code.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  | Type of Code |  |
|  | A program written in the interpreted language BASIC. |  |  |
|  | The program that is the interpreter for BASIC programs. |  |  |
|  | A program written in the Java language before it is compiled. |  |  |
|  | A compiled Java program. |  |  |
|  | The program that is the Java Compiler. |  |  |
|  | The program that is the Java Virtual Machine. |  |  |
|  | A program written in the C++ language before it is compiled. |  |  |
|  | The program that is the C++ Compiler. |  |  |
|  | A compiled C++ program. |  |  |
|  |  |  |  |

🏆 🏆 8. The third video linked earlier discusses how *Just-In-Time (JIT) compilation* can improve the performance of the hybrid approach to program execution. JIT compilation has an interesting relationship to the principles of spatial and temporal locality that we learned about when discussing processor cache. Describe in a few sentences how the performance improvements obtained by virtual machines that use JIT compilation are also dependent upon the principles of special and temporal locality.

**A Container for Program Interpretation and Translation:**

Our classes and labs over the next few weeks will focus on the topic of program interpretation and translation. To do so we will use an *assembler* and a *machine simulator* for a machine that is similar to the Knob & Switch but that is more full-featured. We will use the assembler to translate programs from human readable source code into binary machine language for the machine simulator. We will then be able to run, test and debug these programs using the machine simulator. In addition, in lab you will be writing an interpreter for a simple programming language.

Rather than having everyone try to install and run the necessary tools and programs on their own machine, we will be using a virtual machine that runs in a Docker Container. This section will have you create and test that virtual machine. That way it will be ready for the next homework assignment.

The exercises below assume that you have Docker Desktop installed and working on your machine. If you do not you can use one of the lab machines for this work, or you use the installation instructions for Docker Desktop to install it on your machine:

* For Windows:
  + Install the Windows Subsystem for Linux (WSL) using these directions:
    - <https://learn.microsoft.com/en-us/windows/wsl/install>
  + Install Docker Desktop using these directions:
    - <https://docs.docker.com/desktop/install/windows-install/#install-docker-desktop-on-windows>
  + Download and install the TigerVNC viewer:
    - <https://sourceforge.net/projects/tigervnc/files/stable/1.12.0/vncviewer64-1.12.0.exe/download>
* For MacOS:
  + Install Docker Desktop using these directions:
    - <https://docs.docker.com/desktop/install/mac-install/>
  + Download and install the TigerVNC viewer:
    - <https://sourceforge.net/projects/tigervnc/files/stable/1.12.0/TigerVNC-1.12.0.dmg/download>

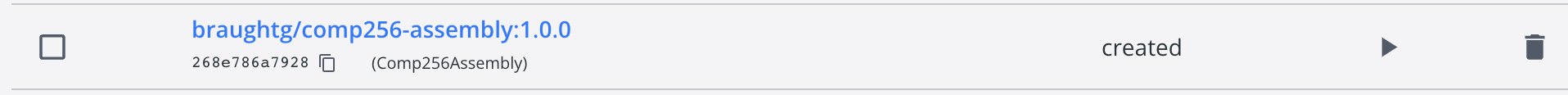
9. Run the Docker Desktop application (find it and double click its icon). Give Docker Desktop a little time to start.

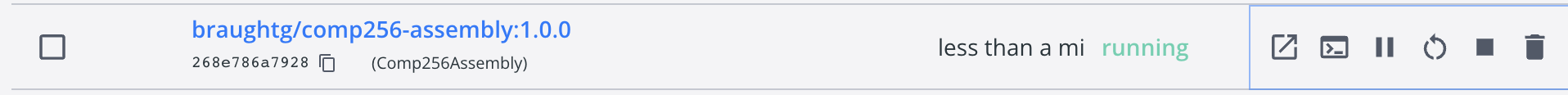
10. Open a Terminal on your machine:

* For Windows: Run the WSL app from the Start Menu.
* For Mac: Run the Terminal app from the Utilities folder in the Applications folder.

11. Copy and paste the following command into the terminal and press Enter or Return:

docker create --name Comp256Assembly --publish 5901:5901 --publish 6901:6901 braughtg/comp256-assembly:1.0.0

12. Using the Docker Desktop application, find the “Containers” tab. Find the entry for the Comp256Assembly container. On MacOS this entry looks as follows:  
  


13. Click the “Play” button () to start the container. The container should begin running:  
  


14. Open the TigerVNC application that you installed earlier and fill in the address of the “VNC server” with localhost:5901 as shown here:

Graphical user interface

Description automatically generated

15. Click the “Connect” button. When you do you should see a window like the following:

A screenshot of a computer

Description automatically generated

16. The window above is a virtual machine running the Linux operating system and you will use it for the activities and labs in the Language Abstractions unit. To test that everything works, open the terminal by clicking its icon in the “Launcher” at the bottom of the screen.

*Nothing is required here, but you must have a Terminal window open.*

17. When the terminal window opens, enter the command machine and press Enter or Return. The machine simulator should open in a window. Paste a screenshot of that machine emulator window here to show that you have been able to run it.

18. Each time you finish working with this virtual machine you will need to stop the container (like shutting down your computer). To do so, click the “Stop” button () for the container in Docker Desktop. What happens to the TigerVNC window after you click the “Stop” button?

19. After stopping the container you can notice that it remains in Docker Desktop with the status “Stopped.” To use the container again:

* Ensure that Docker Desktop is running (as in step 9)
* Start the container (as in step 13)
* Connect to the container (as in steps 14 and 15).

Notice specifically that you do not need to use the long “docker create” command each time you want to use the container.

Optional: To help me improve and scope these activities for future semesters please consider providing the following feedback.

a. Approximately how much time did you spend on this activity outside of class time?

b. Please comment on any particular challenges you faced in completing this activity.