Heriot-Watt University

Creating a Visualiser Software for the PushGP Algorithm

Author:
Derek Wong

Supervisor:
Dr. Michael Lones

A thesis submitted in fulfilment of the requirements for the degree of MSc

in the

School of Mathematical and Computer Sciences

August 14, 2022

Declaration of Authorship

I, Derek Wong, declare that this thesis titled, "Creating a Visualiser Software for the PushGP Algorithm" and the work presented in it is my own. I confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: Derek Wong 

Date: 14-08-2022

Contents

1 Abstract

The Push language is a type of genetic programming language initially introduced by Lee Spector as a more robust and automatic alternative to the more traditional genetic languages. One of the ways it achieves this is through the implementation of a stack-based system. Due to it still being a relatively new language, coupled with the nichness of stack-based genetic programming, there are not many visual demonstrators for Push. I think a good way to spread awareness of Push is through creating a visualiser which is able to quickly and effectively educate users the basic operations of Push, therefore lowering the initial barrier of entry. I plan on developing a visualiser for Push by looking into the various techniques used within the field of Algorithm Visualisation to increase pedagogic effectiveness, and outline the general features of my visualiser which will be developed within the course of three months. The visualiser will be a free and open-source software aiming to help students to grasp the basics of Push through actively encouraging them to interact and experiment with the language.

2 Acknowledgements

I would like to thank my project supervisor, Dr. Michael Lones for all the continuous support and insightful feedback.

3 Introduction

Genetic programming (GP) is a branch of artificial intelligence, in other words, a computation technique that allows computers to solve problems automatically. GP is systematic, domain-independent, and aims to solve problems from a high-level statement of what needs to be done. It is achieved by "evolving" a population of computer programs through a set of stochastic transformations into a new, and hopefully better population of programs. While the randomness of the process cannot guarantee optimal results, it is certainly able to escape traps that deterministic methods are captured by, and produce novel and unexpected ways of solving problems, not dissimilar to the properties of life which it is trying to emulate [13]. The evolution of a program consists of three distinct operations [6]:

- Selection: The programs which yield the best results will be chosen.
- Crossover: Parts of programs are "cut out" and swapped with each other.
- Mutation: Parts of programs are randomly modified.

While the concept of autonomy is prevalent and very much sought-after within the field of GP, it is rarely fully realised due the fact that the successful application of which requires considerable experience and knowledge, which is antithetical to the main goal of GP - to seek automatic techniques that learn or evolve solutions to problems with minimal human intervention [5].

Given the current state of GP requires a sufficient level of understanding from the user, it would a good idea to start developing ways to effectively teach them to the user. Algorithm visualisation (AV) is a field of study which aims to create technology which graphically illustrates how algorithms work, and in this report we shall review Push3, a type of genetic computing created with autonomy in mind, and attempt to create an application which is able to visualise the Push syntax and computation process through reviewing various techniques use for AV, and their educational effectiveness.

4 Push GP

"Standard genetic programming is not designed to handle a mixture of data types"[9].

One of the most common issues which comes with the traditional parse-tree based GP is the possibility that during the "scrambling" process, the functions may be applied to the wrong data type [5]. Such an error could be potentially catastrophic given the hierarchical structure of the parse-tree - if one function fails to operate properly, then all the functions above it may also produce erroneous results.

A popular strategy to prevent such a problem from occurring is called "strongly typed genetic programming", of which the programmer must define the types of all values, as well as the input and output of all functions. On top of which the evolutionary side (program generation, crossover, and mutation algorithms) will also have to be programmed to obey they type restrictions. However, such a strategy will not only complicate the development process, but also could limit the domain and range of the program search space [5].

On the other hand, a stack-based GP is organised in a way that does not nest the return value of a function within the input of another function, instead a function receive its argument from a stack containing data of the appropriate type, and subsequently return the result by pushing it onto the suitable stack [12]. This method makes the GP more robust to evolutionary operations.

4.1 Benefits of Using Push

Tests have shown that the Push language performs better than the Koza-style GP at solving problems with higher arity, though it does tend to under perform when working on simpler problems [5]. In other words, Push has excellent scaling capabilities, and therefore will work best when applied to search spaces with a higher dimensionality.

4.2 Syntax

A Push program consists of a sequence of zero or more instructions and literals, enclosed in brackets whenever necessary. The only syntax requirement is that there are no loose brackets [8]. Here is an example of what a Push program may look like:

```
((:exec_dup(1 2 :integer_add)) :integer_eq)
```

As we can see, the instructions always begin with a colon, and the literals present are 1 and 2. This code is fully ready to be pushed onto Push3's EXEC stack.

4.3 Executing a Push Program

The execution (EXEC) stack is a new addition to Push3, and is responsible for storing the Push program to be sequentially executed by the interpreter. As long as the stack is not empty, it will pop the topmost element (E) and process it [8].

If E is a literal, then it will be pushed onto the appropriate stack, Push provides multiple stacks for literal storage, one for each data type. Referring to equation (4.2), the 1 and 2 will be pushed onto the INT stack when they are popped from the EXEC stack.

If E is an instruction, then it will be executed by popping any required inputs from relevant stacks, perform computation with the popped data, and pushing any outputs to their appropriate stacks. In this case, the `:integer_add` instruction will pop two elements from the INT stack, add them together, and push the output back to the same stack. The types of input and output associated to the instruction are specified during the implementation of said instruction, for the current equation, the data type the function manipulates is denoted on the left side of the underscore, while the function is on the right, for example, the "add" operation exists both as `:integer_add` and `:float_add`, this is because the they manipulate separate stacks. This system allows instructions to always receive inputs and produce

output, regardless of how the rest of code is structured [8]. If the instruction receives insufficient input from the appropriate stacks, then it will be skipped and therefore causes no effect.

It is worth noting that the code itself is also a valid data type. In (4.2), the instruction `:exec_dup` serves to duplicate the code encapsulated by the following bracket, and push the duplication onto the top of the EXEC stack. The ability for a program to alter its own code lends flexibility in regards to control, allowing programs to dynamically create novel control structures and subroutine architectures [8].

4.3.1 Dry Run of the Example

The topmost element in the EXEC stack is the `:exec_dup` instruction, and therefore the following bracket-enclosed code in the stack would be duplicated, resulting in the EXEC stack being in the following state:

```
EXEC: ((1 2 :integer_add) (1 2 :integer_add) :integer_eq)
```

The next two elements to be process are the two literals: 1 and 2. They will be popped from the EXEC stack and pushed onto the INT stack:

```
EXEC: (:integer_add (1 2 :integer_add) :integer_eq)
```

INT: 2, 1

The instruction `:integer_add` will now be processed by taking the top two elements from the INT stack, and pushing the output back onto the INT stack:

```
EXEC: ((1 2 :integer_add) :integer_eq)
```

INT: 3

The next block of bracketed code is identical to the previous, therefore the same result will be produced:

```
EXEC: (:integer_eq)
```

INT: 3, 3

Finally the instruction `:integer_eq`, is performed by taking the top two elements from the stack and check if they are equal in value, then output the result as a boolean to the appropriate stack, producing the final result:

EXEC:

INT:

BOOL: True

5 Algorithm Visualisation

Algorithm Visualisation (AV) aims to facilitate the understanding of how algorithms work through depicting them as either a discrete or continuous sequence of graphics, the viewing of which is controlled by the user.

Due to its versatility, there are many situations within the context of computer science education which can benefit from AV [2]:

- Lecture:
  When teaching an algorithm, a lecturer can employ AV to help better demonstrate the aspects of an algorithm by graphically representing its operations.

- Study:
  It is commonly agreed that encouraging student interaction will have a positive effect on learning [4]. Therefore a good AV application should allow students to experiment with the algorithm such as modifying the input parameters and comparing the output. This type of learning can take place outside of the classroom, which promotes active learning instead of just passively watching a video explaining the algorithm.

- Test:
  There are many ways AV can be used as a test for students: A portion of an algorithm animation can be presented to a student, who is then asked to identify the algorithm; students can also be made to predict the outcome of an algorithm after viewing the first portion of its operation. This could potentially promote critical thinking, remedying the problem known as "hands-on, mind-off", where the application engages with the user in such a way that discourages analytical thought [4].

5.1 Concepts Involved

The way information is presented in the visualiser is the core factor of how effective it will be [14]. Most of the researches into AV have indicated that the educational effectiveness can be attributed to a combination of three different concepts - Epistemic Fidelity, Dual-Coding, and Cognitive Constructivism [2].

5.1.1 Epistemic Fidelity

Epistemology, within the current context, is a framework that assumes that humans contain their own model of the physical world in their minds, which they use as the basis for their logic and reasoning. We then assume that it would be possible for one to attempt to graphically present such a mental model for others to observe and internalise the information encoded within. Therefore if we are able to graphically encode an expert's mental model of an algorithm, we can then efficiently transfer the information contained within to others [2].

This concept is demonstrated in studies by observing that students may understand the algorithm better when it is represented in a certain way, all the while the teachers may prefer a different one [7]. This highlights that one's knowledge and experience may lead them to prefer different types of graphical models, and therefore when creating our application, it is important to identify the intended audience in order to create an educationally-effective graphical model.

5.1.2 Dual-Coding

Our cognition, according to the Dual-Coding Theory, involves the activity of two interconnected by functionally-separate systems - a verbal system which deals with written and spoken languages, and a non-verbal system which handles anything that is non-linguistic in nature, such as imagery and sounds [11]. Visualisations which can transfer information using both systems will allow viewers to form connections between the two systems, and therefore

better understand and memorise the algorithm.

Simple techniques such as labelling can form such a connection between keywords and graphical elements, and is therefore found to be more comprehensible by viewers [7]. It is shown that "text is important for precision, pseudocode is useful for conveying steps of the algorithm, and animations are good for depicting operational behaviour" [4]. In summary, it would be educationally-effective if we are able to simultaneously animate the operations of an algorithm, while being able to highlight important details using verbal descriptions.

5.1.3 Cognitive Constructivism

In Cognitive Constructivism, we work under the assumption that knowledge is not grounded in absolute objectivity, but is rather constructed by individuals, the understanding of which is moulded by their subjective experiences in the world [2]. This implies that knowledge has to be actively constructed by the student, instead of being passively transferred by the teacher. Within the context of AV, it means the more the user interacts with the visualisation, the more they will be able to learn from it.

One of the studies has shown that if a student is unable to properly interact with the visualisation, it is easy for them to become lost during the viewing and quickly lose interest [15]. Therefore an effective visualiser should be able to allow active user intervention, be it altering the speed of the animation, rewinding the operations, or to alter the inputs required by the algorithm. Another study has shown that the educational effectiveness of an AV could be improved further by providing comprehensive motivational instructions alongside the animation [19].

6 Current Visualisers for Push GP

Given that Push is still a rather relatively new language, there are not a whole lot of visualisations to be found. Along with a quick video introduction to Push [17], the only one which is readily available online is the Push Interpreter found on the official website for PushGP [18].

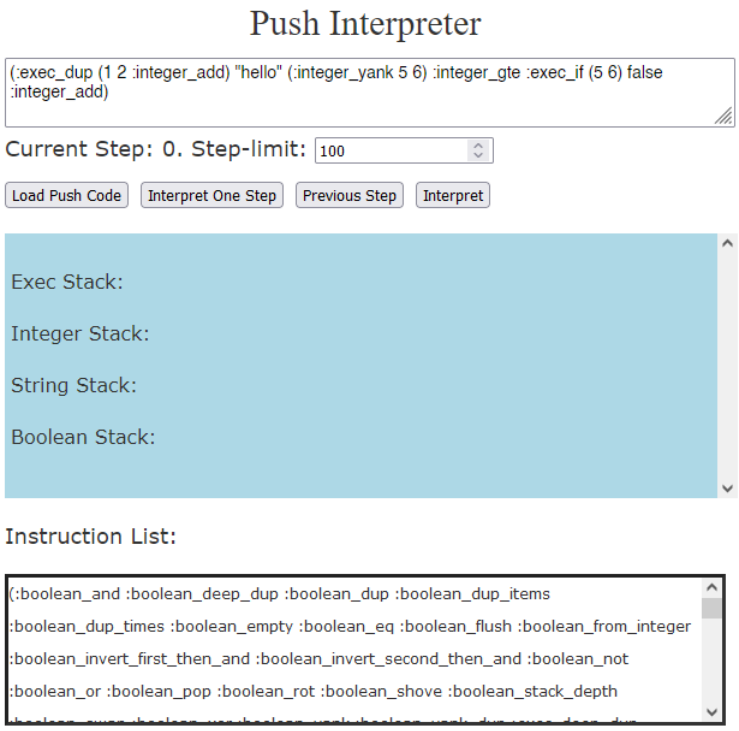

Figure 1: The Online Interpreter for PushGP

In figure 1 we can see how the online interpreter is presented in its entirety. Its functionalities include the ability to load any custom Push code (the code seen in the text box is the one that comes pre-loaded with the interpreter), granted that the given instructions can be individually found within the Instruction List; the user is also able to interpret the program step by step, and is also able to have the interpreter go back a step; if one only wants to view the final output of the program they can do so by simply clicking on the "Interpret" button.

6.1 Critical Analysis of the Existing Visualiser

Even though the Push Interpreter shown in figure 1 does no present itself as a visualiser, but simply as an interpreter, it is still the best/only existing readily-available visualiser for the Push language. If someone would like to start learning and experimenting with the algorithm, they would only have access to a few tutorial videos and this interpreter. As Section 5 has covered that the ability to interact with the visualisation is crucial to the understanding of the algorithm, I shall look into the Push Interpreter's ability to do so by analysing how well it uses the 3 concepts described in section 5:

Epistemic Fidelity:
The interpreter visualises the program being loaded into the EXEC stack from the text box, as well as how different types of data is being moved and manipulated within different stacks. This format very much represents the structure of the language and how it operates.
However none of the changes are animated/highlighted, texts just disappear when the data they represent are processed, and new texts representing the output will then suddenly appear somewhere else. This could lead to the user being lost, losing track of the exact effects of certain instructions, especially if more than one type of data from different stacks disappear at the same time.

Dual-Coding:
Nothing much is done in terms of this concept, besides having the different sections looking visually distinct - The input section being a text box, the stack representations are within a blue box, and the list of instructions are in a box with a bold black border.
The easiest way to implement this concept is to perhaps represent individual types in various colours, and to clearly distinguish instructions from literals. The user will then be able to easily tell what kind of data will be processed next, encouraging them to predict the state of the program in the upcoming step.

Cognitive Constructivism:

The ability to go forward and back a step is an effective feature to encourage interaction and prevent the user from getting lost. The "Interpret" button is also useful if the user would like to perform a dry run of a program themselves and see if their prediction of the final output is correct.

The biggest problem is that there are no easy ways to tell what each instruction do, besides running them and try to guess based on the input taken and output produced. Therefore it may be difficult for the user to experiment with the instruction which they are given.

7 Requirement Analysis

The following section will detail the goals to the project, what I plan on delivering as a final product, the stakeholders of this project, as well as what my priorities are when developing the visualiser.

7.1 Project Goal

The overarching goal of the project is to create a visualiser for the Push language. The visualiser should be educationally effective, meaning that the user should be able to grasp the basic syntax and operations of Push just by interacting with the visualiser alone, without having to consult additional sources of information such as papers or video guides. The target audience for this visualiser will be high school/university students who already possess a certain level of affinity with programming languages, and the visualiser will be designed based on that aforementioned group of audience in mind.

7.2 Deliverables

Currently the plan is to create a standalone application as an executable file which will be designed to run on a Windows machine. While being less accessible than having it being able to run on a website, the fact that it can be stored locally on a machine, and does not require internet connection to run is a big plus.

I plan on developing the visualiser using Unity, a free-to-use game engine well known for its accessibility. Given that the visualiser will heavily depend on presentation, namely animation and user interface, which are one of the many features Unity excels at.

7.3 Stakeholders

There are several groups which could benefit from the success of the project, the requirements for success for each of them are as follows:

Stakeholders	**Requirements**
Project Author	Complete Project
Project Superviser	Well-Executed Project
Heriot-Watt University	Well-Executed Project
Bio-Inspired Computation Students	Well-Designed Visualiser
Bio-Inspired Computation Teachers	Well-Designed Visualiser

7.4 Priorities

In order to complete this project successfully, I must first identify the focus and limits of it. The following section will describe the various components of the project, and for each component, I will detail the features which will absolutely be included, the functions which could be implemented if time allows, as well as areas which will not be covered during the length of the project.

7.4.1 Push Algorithm

In order for the parameters of the visualiser to be fully configurable by the user, I must either write my own Push interpreter program, or adapt one of the existing Push libraries into the visualiser. I will attempt to implement all the instructions available to the online Push interpreter.

While Push is a type of GP, I will only focus on how the program is parsed and run. I will not attempt to include any genetic operations such as crossover or mutation. This is due to the inclusion of the genetic operations will drastically inflate the scope of the project, as well as distract the focus from the Push algorithm itself. Moreover there are already existing visualisers out there which are capable of demonstrating them [3].

7.4.2 User Interface

The design of the user interface will play a major role in determining the educational effectiveness of the visualiser, therefore it is very important that I spend a lot of time on its development.

Each instruction and data type much be distinct from each other, so that the user can tell them apart, the most straightforward way of doing this is to have them be in different fonts/colours.

The operations performed by the EXEC stack must be clear to the user, this will be done through animating any changes. The properties of instructions and literals must also be well labelled so that the user can try to predict the outcome of the next step.

In order to demonstrate Push's robustness, the user should also have the ability to alter the order of the program within the EXEC stack. While it is technically achievable by just altering the program within the input text box and re-parsing it, being able to simply click and drag will prove to be more intuitive, and will encourage further user interaction.

The ability to go a step forward and back will definitely be included in the visualiser, along with relevant buttons and animations. The user can also choose to watch the whole program play out with a single button press, or to even skip the animations and jump right to the final outcome, this will allow users to quickly confirm their own guesses of how the program will run.

7.4.3 Additional Educational Features

If given enough time, I would also like to implement a quiz function to the visualiser to further enhance its educational capabilities. An example of a quiz feature is to have the visualiser randomly generate a program and calculate its outcome, then "scrambles" the instructions and literals of the program before showing both the scrambled program and the outcome to the user. The user is then tasked to rearrange the elements in order for it to yield the expected outcome.

7.4.4 Multi-platform Support

Unity is known for its ability to develop on various platforms, namely Windows and Mac. The visualiser will be developed with the intention of it running on Windows. While I will attempt to build a Mac version upon the completion of the visualiser, though if it does not run the way it should, I have no intentions of fixing it.

7.4.5 MoSCoW Classification

The following table contains a list of priorities from the descriptions above using the MoSCoW method [10]:

Table 1: MoSCoW Table

Requirements	**Priorities**
Push Algorithm	
Contains all instructions from the online interpreter	Must
Ability to parse custom program	Must
Ability to execute custom program	Must
Capable of performing genetic operations	Won't
User Interface	
Animated operations	Must
Visually-distinct program elements	Must
Well-written description for each instruction and data type	Must
Ability to rearrange elements within the EXEC stack	Should
Time controls such as forward, rewind, and skip to end	Must
Additional Educational Features	
Automated quiz function	Should
Multi-Platform Support	
Windows build	Must
Automatic MacOS build	Should
Manual port to MacOS	Won't

8 Implementation Plan

8.1 Existing Push Libraries

One of the core features of my visualiser is the ability to convert user input into instructions for the visualiser, which means I will have to implement an interpreter, be it from programming one myself, or finding one online. There are many existing PushGP libraries ready to be used on the website in various programming languages. It may save me some time to try to use one of them instead of programming an interpreter from scratch myself. Given that I will be using Unity, which uses C# to handle its scripting functions, it is worth looking into PshSharp, a C# implementation of PushGP [16].

However, unlike the goal of the visualiser, the libraries out there intend to provide all the in-depth features of Push, meaning to actually be a functional GP instead of simply being a demonstrator. This could potentially mean that more time will be spent trying to tweak and optimise the library to fit the project.

Therefore I currently plan on writing a simplistic Push interpreter myself, only including functions and features that would be used for the visualiser.

8.2 Product Use-Cases

8.2.1 Typical Use-Case

The most basic scenario for the visualiser is depicted in figure 2, where a user loads up some instructions from the input box, and is then able to see the visualised program in action, along with being able to go back and forth using available buttons. The user will also be able to reset the visualiser at any point to adjust the inputs.

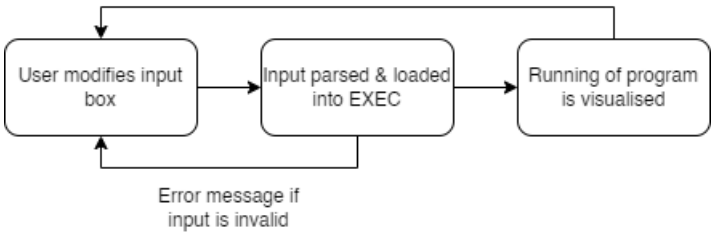

Figure 2: Typical Use-case

8.2.2 Quiz Mode

In order to be educationally effective, I intend on implementing a quiz mode to the visualiser for the user to test out their understanding of the algorithm. The current implementation plan is to have the visualiser generate a program consisting of a random number of instructions and literals, it then calculates the outcome of the program and presents it to the user along with a "scramble-up" version of the program. The user is then expected to reorder the elements in the program in order to have the final result match up with the original one. Figure 3 outlines the steps taken to generate a quiz.

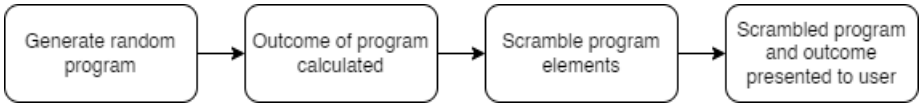

Figure 3: Quiz Generation

8.3 Interface Design

I have drawn out a simple wireframe design for how the visualiser would look like. Further explanations will be given to each UI element down below. Note that this is simply a general guideline which I will follow, and it is likely for changes to the position and size of each UI element to occur.

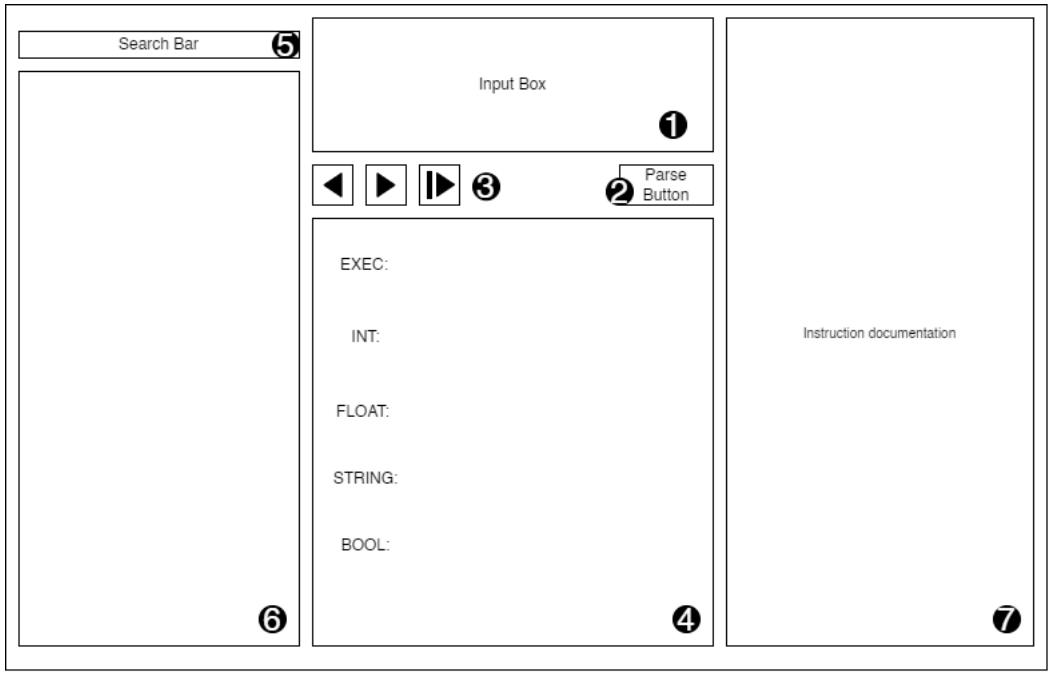

Figure 4: Wireframe

Table 2: UI Details

Label	**Name**	**Function**
1	Input Box	User can type in their custom programs here. Visualiser initiates with a pre-written program.
2	Parse Button	Loads the program in the Input Box to the EXEC stack
3	Time Control Buttons	Allows the user to control the visualisation.
4	Stack Display	Displays the current state of the stacks
5	Search Field	Allows the user to look up available instructions
6	Search Results	Displays instructions which matches the user's search queries
7	Instruction Documentation	Displays a detailed explanation of the function the user clicks on

8.4 Milestones

In this section I will plan out a rough timeline ordering the sequence which the goals should be achieved.

1. **Interpreter and Syntax Checking**

   I will write a program which is able to read user programs and execute them, no visualisation features thus far. This means the implementation of the algorithm, as well as a wide range of instructions for the program to execute.

2. **UI Design and Animation Implementation**

   I will begin the design of UI features, and integrate it with the previously-written program. I will then create and implement animations for the visualiser. After achieving this milestone, the visualiser should be able to perform its most basic function, which is to visualise the execution of a Push program.

3. **Instruction Documentation**

   It is important for the user to know what available instructions there are in the visualiser, as well as what they do. I will create documentation for each instructions, as well as implementing a way for the user to search for them.

4. **Quiz Features** Additional features needs to be implemented in order for the proposed Quiz function to work. They will be implemented at this point in time..

9 Testing and Evaluation

Besides trying to meet the requirements established in previous chapters, human evaluation is also required in order to to assess whether or not the project is a success. In this section, I will detail my plans to recruit volunteers for the testing and assessment of the visualiser, along with the surveys which will be used to standardise the feedback.

9.1 Evaluation Plan

The main point of the evaluation is to assess the effectiveness of my visualiser at educating the user about the basics of the Push algorithm. Moreover, it would be productive to receive feedback from experts within the field (e.g. bio-inspired computing professors) on the accuracy of the visualisation.

The recruitment process starts with me making contact with people of interest, either through email or other messaging software. Within the message I will attach relevant links to the project, namely the documentation of the visualiser, to provide further information. Interested participants will be sent a survey, assessing their knowledge regarding basic mathematics, computer science, and bio-inspired computation. After which, they will be given a quick quiz on the syntax and algorithm of PushGP, upon completing the quiz, the participants will be given access to the visualiser and are asked to familiarise themselves with PushGP through nothing but the visualiser. In a few days, I will once again send them a PushGP quiz with different content but of similar difficulty to the first one. The results of the first and second quiz will then be compared, and I will be able to evaluate the effectiveness of the visualiser. Finally, I will ask the participants to fill out a System Usability Scale (SUS) survey, which I will detail in the secion below.

9.2 System Usability Scale

The System Usability Scale (SUS) is a "reliable, low-cost usability scale that can be user for global assessments of the systems usability" [1]. The user is presented a list of questions regarding their experience with the system, and is asked to provide one of the five ratings (strongly agree, agree, neutral, disagree, strongly disagree) to each question. The 10 questions included in paper [1] are:

1. I think that I would like to use this system frequently
2. I found the system unnecessarily complex
3. I thought the system was easy to use
4. I think that I would need the support of a technical person to be able to use this system
5. I found the various functions in this system were well integrated
6. I thought there was too much inconsistency in this system
7. I would imagine that most people would learn to use this system very quickly
8. I found the system very cumbersome to use
9. I felt very confident using the system
10. I needed to learn a lot of things before I could get going with this system

10 Project Plan

In this section, I will assess the possible risks in the project, as well as create a estimated schedule for each task to be completed for the project.

10.1 Risk Assessment

Risks will always be present in any kind of projects, therefore it is important to identify them and come up with methods to help either mitigate or avoid them.

The first and possibly the most noteworthy one which comes to mind is the risk of losing all the data and having to start the project from scratch. This can be prevented, if not largely mitigated by having both local and online backups of the project.

Other risks which could potentially prevent the project from being completed on time could include mistakes such as poor planning or bad coding, which may create extra work for myself. Unpredictable events such as illness or hardware failure should also be accounted for. The table below will detail the likelihood of each risk, as well as action taken to mitigate them.

Table 3: Risk Assessment

Risk	**Likelihood**	**Impact**	**Mitigation**
Personal Illness	Low	Medium	Continue with lighter workload Reschedule future tasks Apply for mitigating circumstances
Data Loss	Low	High	Create regular local backups On both local machine and GitHub
Hardware Failure	Medium	Medium	Perform maintenance tasks Replace faulty computer parts
Bad Implementation	High	Medium	Search for solution online Ask supervisor for help Find a work-around
Behind Schedule	Medium	Medium	Give generous margin Redesign project scope

11 Task Scheduling

There will always be a degree to inaccuracy when it comes to estimating the total time needed to complete a project, therefore I will break it down into a list of tasks of varying difficulties: the easiest tasks will be assigned three days each to complete, the slightly harder ones will take a weekday's worth, and the hardest will take 10 days to account for unknown issues which will most likely arise.

Table 4: Schedule

Begin	**End**	**Task**	**Difficulty**
01-May	03-May	Learn PushGP basics	Easy
04-May	13-May	Create interpreter framework	Hard
14-May	18-May	Implement basic instructions	Medium
19-May	21-May	Basic GUI design	Easy
22-May	24-May	GUI integration	Medium
25-May	04-Jun	Animation design	Hard
05-Jun	09-Jun	Further instruction implementation	Medium
10-Jun	12-Jun	Add search bar to GUI	Easy
13-Jun	22-Jun	Add documentation display	Hard
23-Jun	01-Jul	Write documentation for instructions	Hard
02-Jul	11-Jul	(Optional) Quiz mode	Hard
12-Jul	21-Jul	(Optional) Drag and drop feature	Hard
22-Jul	26-Jul	Begin recruiting	Medium
27-Jul	29-Jul	Send questionnaire and quiz	Easy
30-Jul	4-Aug	Send visualiser	Medium
5-Aug	7-Aug	Send second quiz and SUS	Easy
8-Aug		Dissertation redaction	

12 Professional, Legal, and Ethical Issues

This project does not concern any deep ethical issues. However certain points will have to be addressed, these are general concerns such as the choosing of participants for their feedback and the protection of their personal data, as well as the open-source model the project should follow.

12.1 Participant Recruitment and Consent

I will attempt to recruit participants through either email or other messaging systems. The potential participants will be informed of the details of the project, as well as their role within it. Whether they choose to take part in the project would be up to themselves, they will also be free to withdraw from the project at any time without consequences.

12.2 Protection of Personal Data

The consent form for the project will ask for the name and contact methods of the participants so that they can be easily identified from the group and made contact with during the process, these data will not be published. Moreover the participants can choose to use pseudonyms to avoid being personally identified.

12.3 Open-Source Model

The end-product of this project will be a free learning software which aims to effectively visualise the Push GP algorithm. The compiled executable will be uploaded onto GitHub along with the source code, should anyone want to look at how the visualiser functions, or to add additional features. Given that I will be developing the visualiser using Unity, I will be using its Personal license [20], which allows me to distribute and modify the project files as I wish.

13 Implementation

13.1 Visualiser Design

The final appearance of the Visualiser does not deviate much from the initial design (Figure 4). A screenshot of the main page is shown below:

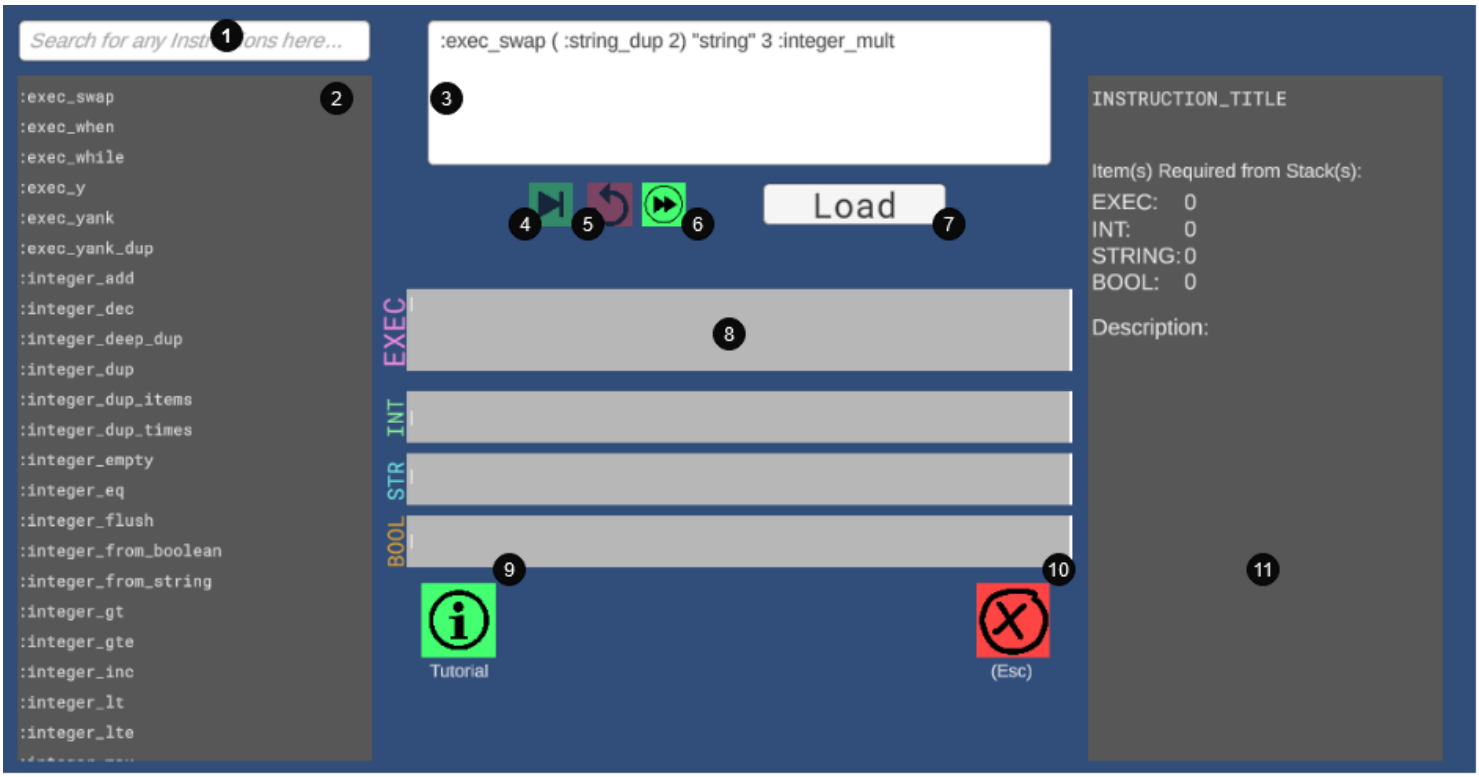

Figure 5: Visualiser

13.2 List of Implemented Features

1. **Search Bar**
   Allows the user to look up specific instructions.

2. **Instructions List**
   Provides a comprehensive list of instructions that are included in the Visualiser, the Search Bar can be used to narrow down the list.

3. **Input Textbox**
   The place where the the user can write their own program and have it be loaded into the Visualiser. A default program is provided to act as an immediate example of how a Push program works; it also gives the user something to experiment with without making them have to come up with a program on their own, which could prove to be difficult when one is just starting to learn the language.

4. **Forward Step Button**
   Runs the program by a single step. Any items popped from stacks will quickly fade away, instead of outright disappearing, this gives the user an impression of which items are popped. If items are generated from the step, they are spawned at the top of the stack and quickly 'falls' to the bottom. The button is disabled during the process.

5. **Undo Button**
   Reverts the most recent step of the program. Unlike the Forward Step button, no animation is played during the process due to limitations of how it is coded.

6. **Fast Forward Button**
   Unlike its counterpart in the Interpreter, which immediately calculates the end result of the program, the FF button effectively moves the program a step forward as soon as the previous step finishes. This means all the animation involved in the steps will still be played out. While being a lot slower than the online interpreter at producing an end result, this method will be clearer at demonstrating to the user

how the end result is produced. This method of implementation also allows the user to stop the fast-forwarding process at any time.

7. **Load Button**
   Initially named the "Parse" button, but was changed later due to some users not being familiar with the terminology. This button will attempt to convert what was written in the Input Textbox into either instructions or literals, and pushes them individually into the Exec Stack.

8. **Stack Displays**
   Whenever a new item is pushed onto any stack, it will appear on the far left, and quickly moves its way to the right, stopping just before the previous top item in the stack, or the right edge of the display.

9. **Tutorial Button**
   The Visualiser comes with a set of simple tutorials to introduce the basic concepts of a stack-based programming language, they roughly outlines the definitions and properties of stacks, instructions, and types. Examples of each are also included, which are akin to loading the default program in the Input Textbox.

10. **Quit Button**
    The Visualiser will attempt to open in fullscreen mode. In order to quit, the user can either click on the Quit Button, or press the escape key on their keyboard.

11. **Instruction Display**
    Whenever the user clicks on an instruction in the Instruction List(2), the properties of the instruction will be displayed. It includes the number of parameters required from each stack, as well as a brief description of its function.

13.3 Concepts Involved

In this section I will refer back to the concepts mentioned in 5.1, and explain how the Visualiser effectively uses them in order to better educate the user.

- **Epistemic Fidelity:**
  Items are now animated when they are either pushed or popped from a stack. Doing so will better convey to the users the "first in, last out" nature of stacks.

- **Dual Coding:** Each item types are distinctly colour-coded, making them easier to spot when there are a lot of items in the Exec Stack.

- **Cognitive Contructivism:** The pre-written program allows the user to immediately experiment with the language by perhaps changing its order or altering the values of the literals. This is much more effective than having just an empty box because it requires much less from the user. The examples in the tutorial section should also provide the same function.

  Having the list of instructions as well as short descriptions of their function right next to the Visualiser also encourages active exploration without having the user switch out to the web-browser to look up the various functions.

14 Evaluation

14.1 Methodology

The most straight-forward way to assess and compare the educational effectiveness of the Visualiser as well as the Interpreter is to quiz the user after having them experience both programs. I have composed two separate quizzes of equal difficulty consisting of 10 questions each. The user will then complete one after using the Interpreter, and complete the other after using the Visualiser. After gathering enough data, I will compare the average score achieved in the two quizzes. If there is an improvement in the score after the Visualiser has been used, then it should suggest the effectiveness of the Visualiser. A System Usability Survey will also be sent to the user when they finish using the programs in order to assess the user-friendliness of the Visualiser comparing to the Interpreter.

14.2 Quiz Design

The quiz format does not deviate much from how the Interpreter/Visualiser is presented. The user will be shown a Push3 program, and is asked to find out what the end result will be by doing a dry run. It is important to include all the basic key concepts involved in learning Push3, which I have broken down into the following:

- **Brackets**
  Certain instructions which manipulate the Exec stack will treat all items enclosed in a single bracket as a single item. An example question would be to ask what item(s) are duplicated by the `:exec_dup` instruction when the Exec stack is in the following state:
  `:exec_dup (1 2) 3`
  Given that the `:exec_dup` instruction will duplicate the next item in the stack, the item duplicated would be `(1 2)`.

- **Invalid Operation**
  When an instruction is executed with insufficient parameters, it will simply be popped from the Exec stack without actually doing anything. An example question would be to ask what the top item is in the Boolean stack after the following items in the Exec stack are executed:
  `True :boolean_and`
  Since the `:boolean_and` instruction requires 2 items in the Boolean stack in order to run, it would be removed from the Exec stack, the top and only item in the Boolean stack remains to be "True".

- **Stack-Based Operations**

  Questions which touch on the "first in, last out" concept will be asked. An example would be to ask for the output of the `:integer_gt` instruction when the following items in the Exec stack are executed:

  `1 2 :integer_gt`

  The `:integer_gt` instruction takes the top 2 items from the Integer stack and compares them, if the second item is greater than the first, it will return True to the Boolean stack, and False if otherwise. In this case, since the 1 is the first item to get pushed to the Integer stack, it would be the second item in the stack, while 2 will be the top item. Therefore when the instruction is executed, it will return False to the Boolean stack.

- **Item Types**

  It is important to know how Push3 recognises the different item types - A string will be enclosed in quotes, while an instruction will start with a colon. Users will be asked to distinguish the difference between literals such as 42 and "42", the former being an integer and the latter being a string.

- **Instruction Memorisation**

  Given that most questions asked in the quiz will involve an instruction or two, it would be unreasonable to require the user to memorise all the instructions included in the quiz. Therefore for most questions, I will include a brief description of the instruction involved. Though for certain instructions, I will request the user to actively seek out their function when they have access to the Interpreter/Visualiser. These instructions will not be accompanied by a description when they are presented in the quiz, and thus requires the user to memorise them. This will hopefully indicate how effective each software is at making the user retain the knowledge of the instructions.

14.3 Results and Analysis

I have gathered 10 participants in total through online messaging, and put them through the aforementioned evaluation method. The results gathered are as follows:

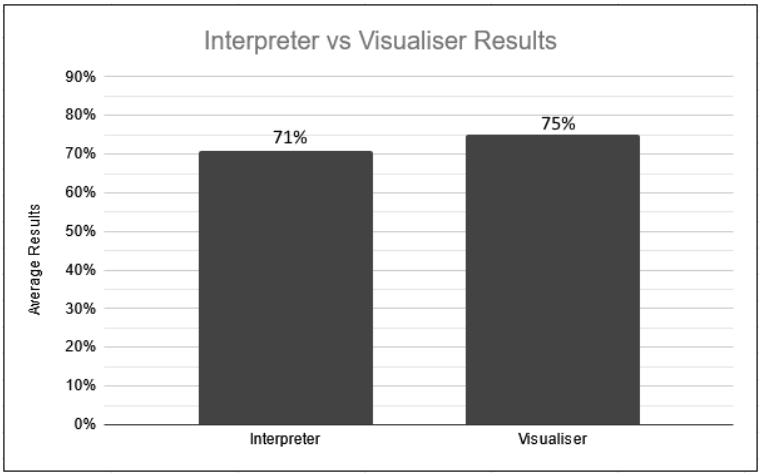

Figure 6: Average Score

In the Interpreter quiz, the users often had trouble grappling with the concept of stacks. I have also received comments from users during the testing process that they do no know how to get certain commands to work, this is most likely due to the fact there is no way to find out the parameters required for instructions beside trial and error.

The trend continues in the Visualiser quiz, where most of the scores lost are also due to misunderstanding the functions of instructions. However questions involving stack ordering are answered correctly.

While there is a marginal increase in the average scores achieved after using the Visualiser, one could argue that this is simply due to the user being relatively more familiar with the system after using the Interpreter. However does show promise, given that there are still a lot more features which can still be implemented to the Visualiser.

14.4 User Feedback

The results from the System Usability Survey has shown that the users have a mixed response to the Interpreter, stating that it is hard to learn as well as being cumbersome to use. The responses to the Visualiser, on the other hand, have been mostly positive. I have also received comments from certain users that the Visualiser is a lot better suited for education.

I have also received a good deal of suggestions regarding what extra features the Visualiser could have in the future, such as:

- Ability to look up an instruction by clicking on it in the Stack Display.
- Ability to highlight and copy instructions when they are displayed in the Instruction Display.
- More examples, ideally an example for each instruction.
- Ability to reorder items in the Stack Display via drag-and-drop.
- Bigger Exec Stack to hold more items.

15 Conclusion and Future Works

Given that the main goal of the study is to create a Visualiser to introduce and educate users on the basics of the Push3 language, the software created and results gathered does help to pave the way to further developments and studies. It has also filled a niche within the existing Push3-related projects.

More work can definitely be done regarding the matter, such as further development of the Visualiser by perhaps implementing the user suggestions stated above. A better-conducted evaluation will also be conductive to assessing the effectiveness of algorithm visualisation. One clear improvement would be to seek out 2 separate groups of people to test out the Interpreter and the Visualiser, instead of having the same group test out one after the other.

References

[1] John Brooke. "SUS: A quick and dirty usability scale". In: *Usability Eval. Ind.* 189 (Nov. 1995).

[2] John T. Stasko Christopher D. Hundhausen Sarah A. Douglas. "A Meta-Study of Algorithm Visualization Effectiveness". In: *Journal of Visual Languages & Computing* 13 (2002), pp. 259–290. DOI: 10.1006/S1045-926X(02)00028-9.

[3] Humera Farooq et al. "An interactive visualization of Genetic Algorithm on 2-D graph". In: Aug. 2011, pp. 144–151. DOI: 10.1109/COGINF.2011.6016133.

[4] S. Hansen and N. Narayanan. "Helping learners visualize and comprehend algorithms". In: *Interactive Multimedia Electronic Journal of Computer-Enhanced Learning* 2 (Jan. 2000).

[5] Alan J. Robinson L. Spector. "Genetic Programming ans Autoconstructive Evolution with the Push Programming Language". In: *Genetic Programming and Evolvable Machines* 3 (2002), pp. 7–40. DOI: 10.1023/A:1014538503543.

[6] Annu Lambora, Kunal Gupta, and Kriti Chopra. "Genetic Algorithm-A Literature Review". In: *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*. 2019, pp. 380–384. DOI: 10.1109/COMITCon.2019.8862255.

[7] Andrea Williams Lawrence. *Empirical Studies of the Value of Algorithm Animation in Algorithm Understanding*. UMI Order No. GAX94-08921. USA: Georgia Institute of Technology, 1993.

[8] Maarten Keijzer Lee Spector Jon Klein. "The Push3 Execution Stack and the Evolution of Control". In: (2005). DOI: 10.1145/1068009.1068292.

[9] David J. Montana. "Strongly Typed Genetic Programming". In: *Evolutionary Computation* 3 (1995), pp. 199–230.

[10] *MOSCOW PRIORITISATION*. Apr. 2022. URL: https://www.agilebusiness.org/page/ProjectFramework_10_MoSCoWPrioritisation.

[11] Allan Paivio. "Dual Coding Theory and Education". In: *Pathways to Literacy Achievement for High Poverty Children* (2006). DOI: 10.1007/BF01320076.

[12] T. Perkis. "Stack-based genetic programming". In: *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*. 1994, 148–153 vol.1. DOI: 10.1109/ICEC.1994.350025.

[13] Nicholas F. McPhee Riccardo Poli William B. Langdon. *A Field Guide to Genetic Programming*. 2008. ISBN: 9781409200734.

[14] Stephen Rayner Richard Riding. *Cognitive Styles and Learning Strategies*. 1998. ISBN: 9781315068015.

[15] Thomas L. Naps S. Grissom Myles F. McNally. "Algorithm Visualization in CS Education: Comparing Levels of Student Engagement". In: (2003). DOI: 10.1145/774833.774846.

[16] shanecelis. *PshSharp*. Apr. 2022. URL: https://github.com/shanecelis/PshSharp/.

[17] Lee Spector. *An Even Quicker Introduction to the Push Programming Language*. Apr. 2022. URL: https://www.youtube.com/watch?v=VGJWlSC0gl4.

[18] Lee Spector. *Evolutionary Computing with Push*. Mar. 2022. URL: https://faculty.hampshire.edu/lspector/push.html.

[19] John Stasko, Albert Badre, and Clayton Lewis. "Do Algorithm Animations Assist Learning? An Empirical Study and Analysis". In: *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*. CHI '93. Amsterdam, The Netherlands: Association for Computing Machinery, 1993, pp. 61–66. ISBN: 0897915755. DOI: 10.1145/169059.169078. URL: https://doi.org/10.1145/169059.169078.

[20] *Unity Personal License*. Apr. 2022. URL: https://unity3d.com/unity/activation/personal.

16 Appendix

16.1 Visualiser Demo

Youtube link of video demo: `https://youtu.be/bxFdekjTGyI`

16.2 Consent Form

Consent Form

This study aims to evaluate the educational effectiveness of the Push3 Visualisation Program. There are 3 main steps to the evaluation process:

1. A quiz to indicate the educational effectiveness of the current online Push3 Interpreter (10 questions);
2. A quiz to indicate the educational effectiveness of the Push3 Visualisation Program (10 questions);
3. A system usability survey (10 questions).

Participation in the study is completely voluntary and can be withdrawn from at any time, you are also free to ask questions regarding the study throughout the process. No personal data will be collected from the study, you can also request for your data to be sent to you, or be suppressed from the study.

The data collected in this study will only be used for research purposes for a MSc dissertations project at Heriot-Watt University. We will only use your name (can be a pseudonym) to identify the data collected from you. The published results in the study will be completely anonymised.

If you agree to participate, you will be asked to test our software. You will be given a choice between a Windows or MacOS executable to be downloaded and run on your own device. Please note that neither the HWU nor the investigator accept any responsibility for any loss/damage incurred by the use of the software on your device. By entering your name below, you are giving voluntary consent that you have read all the information above, and fully agree to take part in this study.

Signed:

_____

Date:

_____

16.3 Interpreter Quiz

Push3 Visualiser Quiz 1

User will be asked to figure out the following instructions on their own before starting the test:

- **:integer_deep_dup**
- **:exec_dup**
- **:exec_yank_dup**

*Required

1. What is your name? *

2. The Exec Stack currently looks liks so:

   False 1 2 :integer_gt

   What would be the top item in the Boolean Stack after running the program to completion?

   Hint:
   The :integer_gt instruction takes the top 2 items from the Integer Stack and compares them. It returns True to the Boolean Stack if the second item is greater than the first, returns False if otherwise.

3. The Exec Stack currently looks liks so: *

   :exec_dup (1 2 3) "some string"

   What would be duplicated when the :exec_dup instruction is executed?

4. The Exec Stack currently looks liks so: *

   1 :integer_add

   What would be the top item in the Integer Stack after running the program to completion?

   Hint:
   The :integer_add instruction takes the top 2 items from the Integer Stack, then returns their sum.

   _____

5. The Exec Stack currently looks like so: *

   False "It this a string?" "True" 32

   How many String literals are there?

   _____

6. The Exec Stack currently looks like so:

   1 2 3 4 5 :integer_yank_dup

   What would be the topmost item in the Integer Stack once the program is run to completion?

   _____

7. The Exec Stack currently looks like so: *

   1 2 3 4 -1 :integer_deep_dup

   What would be the top item in the Integer Stack once the program is run to completion?

   _____

8. The Exec Stack currently looks like so:

   :exec_rot 1 ("rotate" 2) 3

   What would be the topmost item in the Exec stack once the :exec_rot instruction is executed?

   Hint:
   The :exec_rot instruction rotates the top 3 items in the Exec Stack (A B C -> C A B).

---

9. The Exec Stack current looks like so:

   False True "False" :integer_from_boolean

   What would be the topmost item in the Integer Stack after the program is run to completion?

   Hint:
   The :integer_from_boolean instruction takes the top item in the Boolean Stack and returns its equivalent value to the Integer Stack (Coverts False to 0, True to 1).

---

10. The Exec Stack currently looks like so:

    True True :integer_from_boolean :boolean_and

    What would be the top item in the Boolean Stack after the program is run to completion?

    Hint:
    The :boolean_and instruction takes the top 2 items from the Boolean Stack and returns True if both items are True, or False if otherwise.

---

8. The Exec Stack currently looks like so:

   :exec_rot 1 ("rotate" 2) 3

   What would be the topmost item in the Exec stack once the :exec_rot instruction is executed?

   Hint:
   The :exec_rot instruction rotates the top 3 items in the Exec Stack (A B C -> C A B).

---

9. The Exec Stack current looks like so:

   False True "False" :integer_from_boolean

   What would be the topmost item in the Integer Stack after the program is run to completion?

   Hint:
   The :integer_from_boolean instruction takes the top item in the Boolean Stack and returns its equivalent value to the Integer Stack (Coverts False to 0, True to 1).

---

10. The Exec Stack currently looks like so:

    True True :integer_from_boolean :boolean_and

    What would be the top item in the Boolean Stack after the program is run to completion?

    Hint:
    The :boolean_and instruction takes the top 2 items from the Boolean Stack and returns True if both items are True, or False if otherwise.

16.4 Visualiser Quiz

Push3 Visualiser Quiz 2

:exec_if :integer_dup_items :integer_stack_depth

1. What is your name?

---

2. The Exec Stack currently looks like so:

   1 True :integer_empty

   What would be the top item in the Boolean Stack after running the program to completion?

   Hint:
   The :integer_empty instruction checks if the Integer Stack is empty. If it is empty, returns True to the Boolean Stack, returns False otherwise.

---

3. The Exec Stack currently looks like so:

   True :exec_if 25 48

   What would be the top item in the Integer Stack once the program is run to completion?

---

4. The Exec Stack currently looks like so:

   :exec_swap (1 3)

   What would be the top item in the Exec Stack once the :exec_swap instruction is executed?

   Hint:
   The :exec_swap instruction swaps the position of the first and second item in the Exec Stack.

---

5. The Exec Stack currently looks like so:

   True "False" "32" 8 "10"

   What would be the top item in the Integer Stack when the program is run to completion?

---

6. The Exec Stack currently looks like so:

   1 2 3 4 2 :integer_dup_items

   How many 2's will there be in the Integer Stack once the program is run to completion?

---

7. The Exec Stack currently looks like so:

   1 2 0 True :string_length :integer_stack_depth

   What would be the top item in the Integer Stack once the program is run to completion?

   Hint:
   The :string_length instruction takes the top item in the String stack and returns its length to the Integer stack.

---

8. The Exec Stack currently looks like so:

   1 4 34 :string_from_integer :string_length

   What would be the top item in the Integer Stack once the program is run to completion?

   Hint:
   The instruction :string_from_integer takes the top item in the Integer Stack and returns it as a string to the String Stack.

---

9. The Exec Stack currently looks like so:

   1 5 "Half A Dozen" :integer_from_string

   What would be the top item in the Integer Stack once the program is run to completion?

   Hint:
   The :integer_from_string instruction takes the top item i the String Stack and attempts to convert it into an integer and return it to the Integer Stack. Returns 0 if the string cannot be converted into an integer.

---

10. The Exec Stack currently looks like so:

    :exec_dup (1 2 ("False")) :integer_stack_depth

    What would be the top item in the Integer Stack once the program is run to completion?

    Hint:
    The :exec_dup instruction duplicates the top item in the Exec Stack.

---

11. The Exec Stack currently looks like so:

    :exec_dup :exec_dup :integer_swap (3 4)

    What would be the top item in the Integer Stack once the program is run to completion?

16.5 SUS

System Usability Scale (Visualiser)

1. I think that I would like to use this system frequently

   Mark only one oval.

   - ◯ Strongly Agree
   - ◯ Agree
   - ◯ Neutral
   - ◯ Disagree
   - ◯ Strongly Disagree

2. I found the system unnecessarily complex

   Mark only one oval.

   - ◯ Strongly Agree
   - ◯ Agree
   - ◯ Neutral
   - ◯ Disagree
   - ◯ Strongly Disagree

3. I thought the system was easy to use

   Mark only one oval.

   - ◯ Strongly Agree
   - ◯ Agree
   - ◯ Neutral
   - ◯ Disagree
   - ◯ Strongly Disagree

4. I think that I would need the support of a technical person to be able to use this system

   Mark only one oval.

   - Strongly Agree
   - Agree
   - Neutral
   - Disagree
   - Strongly Disagree

5. I found the various functions in this system were well integrated

   Mark only one oval.

   - Strongly Agree
   - Agree
   - Neutral
   - Disagree
   - Strongly Disagree

6. I thought there was too much inconsistency in this system

   Mark only one oval.

   - Strongly Agree
   - Agree
   - Neutral
   - Disagree
   - Strongly Disagree

7. I would imagine that most people would learn to use this system very quickly

Mark only one oval.

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

8. I found the system very combersome to use

Mark only one oval.

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

9. I felt very confident using the system

Mark only one oval.

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

10. I needed to learn a lot of things before I could get gonig with this system

Mark only one oval.

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree