

Programming With

LIVE CODE

Community Edition

What can I make with LiveCode?

LiveCode is used to create apps, games, interactive ebooks and comics.



LiveCode is used to create powerful in-house systems and mobile apps for public and private sector organizations.

Higher Computing Science



Materials produced at GHS
By Mr S. Whyte



The Software Development Process

Introduction

The Software Development Process (**SDP**) can be split into **7 main** steps which are carried out in **order**. These steps should be carried out when creating **any** software project and are summarised below.

Analysis



A statement about **what** your program is going to do. You will also include a description of the **main steps** of the problem.

Design



This involves designing both the **user interface** and the **structure** of the **program code**.

We will be using a design notation known as **Haggis pseudocode** to achieve this. More is mentioned about pseudocode on the next page.

Implementation



The implementation stage involves keying in the program code using the built in **text editor** within the programming environment. We will use **LiveCode** to create our programs.

Testing



Testing is an important part of any project. Testing ensures that your program is **reliable** and **robust** in the sense that it should produce the **correct results** and **not crash** due to **unexpected input**.

We should test our program with **three** sets of test data. These are:

- **Normal** (accepted data within a set range)
- **Extreme** (accepted data on the boundaries)
- **Exceptional** (data that is not accepted).



Documentation



Documentation is usually produced in the form of a **user guide** and a **technical guide**. The user guide shows the user how to **use** the **functions** and **features** of the **software** whereas the **technical guide** gives the user information on how to **install** the **software** as well as the **minimum system requirements**.

Evaluation



An evaluation is usually a **review** which shows that your program is **fit for purpose**, in other words, it does **exactly** what it was **designed** to do.

The evaluation should also focus on the **readability** of your program code. For example, if **another programmer** was asked to **maintain** your program code at a later date, would they be able to understand what was going on? You should always ensure your program is **readable** by doing the following:

- **Use of meaningful identifiers** for **variable** and **array names**
- **Use of internal commentary** (*// This subroutine will do the following....*)
- **Effective use of white space** between **subroutines** to **space out** the **program**.
- **Indentation** to show the **start** and **end** of any control structures such as a **fixed loop**.



Maintenance



Maintenance is performed at the **very end** of the project. You will not be required to perform any maintenance on your programs but you will need to know about **Corrective**, **Adaptive** and **Perfective** maintenance. These are covered in the Software Development theory notes.

The Design Process

Pseudocode and Data Flow

The design of a program is **very important** as it allows the programmer to **think** about the **structure** of the program **before** they begin to **create** it.



The most common way to design the logic of a program is to use a text-based notation known as **Pseudocode**. **Pseudocode** is a cross between **programming language** and our own **English language**. It makes a program **easier to understand** without relying on the use of a programs complex **commands** and **syntax**.

The design is built up of two parts, the **first** is the **Stepwise design**. This shows the **main steps** of the program. The **second** part is the **Stepwise Refinement**. This involves **breaking** these **main steps** into even **smaller steps** so eventually, **one line of pseudocode** becomes **one line of program code**.

Here is the program pseudocode to calculate the volume of a room using the variables **RoomLength**, **RoomBreadth**, **RoomHeight** and **RoomVolume**. Study both the **pseudocode** and **data flow** very closely to understand what is going on:

Stepwise Design (the main steps of the program)

- | | | |
|----|------------------------------|---|
| 1. | Initialise variables | No Data flow required |
| 2. | Get room measurements | In/Out: RoomLength, RoomBreadth, RoomHeight |

Data flow explanation for step 3: Since RoomLength, RoomBreadth and RoomHeight will have been initialised to 0 in the subroutine initialise, they are coming into this step with the value 0 and will be given new values according to the size of the room. Hence they are **IN** as 0's and then passed as **OUT** as a new value.

- | | | | |
|----|------------------------------|---|--------------------|
| 3. | Calculate Room Volume | In: RoomLength, RoomBreadth, RoomHeight | In/Out: RoomVolume |
|----|------------------------------|---|--------------------|

Data flow explanation for step 4: The RoomLength, RoomBreadth and RoomHeight variables are passed **IN** to be used in the calculation for RoomVolume. As a result, they are not changing their values so are just passed as **IN's**. The RoomVolume variable would have been set to 0 in the initialise subroutine and as a result of the calculation, RoomVolume will be given a **new** value so it's **IN/OUT**.

- | | | |
|----|----------------------------|----------------|
| 4. | Display Room Volume | In: RoomVolume |
|----|----------------------------|----------------|

Data flow explanation for step 5: Only the RoomVolume is to be displayed and it is **not** changing in value from what was calculated in step 4, so it is just an **IN** variable within this subroutine.

Stepwise Refinement (the main steps further refined into smaller steps)

- | | | |
|-----|---|---|
| 1. | Initialise Variables | Stepwise Refinement: The main steps are broken down further (refined). We use 3.1, 3.2, 3.3, etc. Notice that the pseudocode looks more like our own language rather than that of the programs. |
| 1.1 | SET RoomLength TO (Real) 0.00 | |
| 1.2 | SET RoomBreadth TO (Real) 0.00 | |
| 1.3 | SET RoomHeight TO (Real) 0.00 | |
| 1.4 | SET RoomVolume TO (Real) 0.00 | |
| 2. | Get Room Measurements | |
| 2.1 | SEND ["Please enter the length of the room in metres: "] TO DISPLAY | |
| 2.2 | RECEIVE RoomLength FROM (Real) KEYBOARD | |
| 2.3 | SEND ["Please enter the breadth of the room in metres: "] TO DISPLAY | |
| 2.4 | RECEIVE RoomBreadth FROM (Real) KEYBOARD | |
| 2.5 | SEND ["Please enter the breadth of the room in metres: "] TO DISPLAY | |
| 2.6 | RECEIVE RoomHeight FROM (Real) KEYBOARD | |
| 3. | Calculate Room Volume | |
| 3.1 | SET RoomVolume TO RoomLength * RoomBreadth * RoomHeight | |
| 4. | Display Room Volume | |
| 4.1 | SEND ["The volume of the room is " & RoomVolume & " cubic metres."] TO field "Output" | |

Reminder: What are Variables?

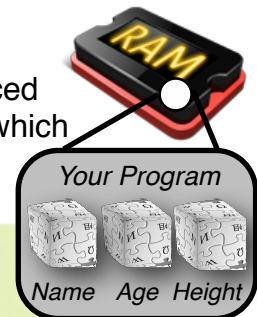
PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE

Let's talk about variables as they are **very** important in programming.

To put it simply, a variable is like a "box" into which data can be placed whilst a program is **running**. We give variables names (identifiers) which suggest or give us a clue as to what data is being held in the variable.

Variables can be store **different types** of **data**, LiveCode supports:

- **Text** (known as **strings**), e.g. *Steven, Jim, or Hello* etc.
- **Real numbers**, (numbers with a **decimal point**) e.g. *3.14, 5.7 or 11.16*, etc.
- **Integer numbers**, (**whole** numbers) e.g. *5, 7 or 102*, etc.
- **Boolean** (**two state** values), e.g. *Yes/No, True/False, 1/0*, etc.



Variables are **identifiers in RAM** used to store **data** in a running program.

Declaring Variables in LiveCode

```
Put 0.00 into RoomLength
// Setup a Real variable called RoomLength and put 0.00 into this variable

Put "" into PlayerName
// Setup a String variable called PlayerName and put "" into this variable

Put 0 into NumberCorrect
// Setup an Integer variable called NumberCorrect and put 0 into this variable

Put False into found
// Setup a Boolean variable called found and put False into this variable

Put True into found
// Setup a Boolean variable called found and put True into this variable
```

Variable Rules

Variables **cannot** contain any **spaces** and must **not** be a **reserved command** in LiveCode. You can tell if a variable has been accepted as it will appear in **black font** when it is typed into the text editor as shown below:

Ask "Please enter the length of the room in metres"

Put it into RoomLength ← This is the variable

In order for the program to know which data is a **variable** and which is **text** to be **printed** in a **put statement**, the ampersand **&** is used.

The ampersand **separates** both the **variable** and the **text** to be printed on the screen. Two ampersands **&&** together will also include a single space when the text is printed. For example the following code:

```
Put "The volume of this room is" &&RoomVolume&& "cubic metres." into field "Output"
```

....will produce:

"The volume of this room is 3000 cubic metres."



Classification of Variables

PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE

Variables fall into **two** main types. The type of a variable determines **where** it can be **used** in a **program**.

The **two** main types of variable are **local variables** and **global variables**. A description of each is given below. It is important that you understand the difference as you will be using **both types** of **variable** in this course.

Local Variables

A **local variable** is one which only exists within **one subroutine, function or procedure**.



Local variables are **created** when the subroutine is called (run) and are then **destroyed** when the subroutine **terminates**. They **cannot** be accessed or assigned a value except within that **subroutine**.

The example below shows the use of a local variable:

```
On Get_Users_Name
// Setup the local variable to be used in this subroutine
Local KeyPressed

REPEAT until KeyPressed = "Y" OR KeyPressed = "y"
  Ask "Please enter the name of the student: "
  Put it into StudentName
  Ask "Are you happy with the name entered? (Y or y for Yes): "
  Put it into KeyPressed
END REPEAT
End Get_Users_Name
```

In the subroutine **Get_Users_Name**, the local variable **KeyPressed** is created. The purpose of this variable is to check whether or not the user is happy with the name that they have entered by keying in "Y" or "y", otherwise the program will keep looping. This local variable is unique to this subroutine and **cannot** be used in any other subroutine.

The **advantage** of using **local variables** is that it **prevents** them from being used **elsewhere** in the program and possibly having their **contents accidentally changed**. It is therefore good practice to make use of local variables in large programs.

Global Variables

A **global variable** is one which can be **accessed** and **altered** from **any** part of the program, even from another script/event so long as it is **declared** at the very **start**.



Global variables should **always** be used with **care** as their values may accidentally change if the programmer forgets that they have already used them in another subroutine. The example below shows the setting up of a series of global variables in LiveCode:

```
// Setup the global variables to be used in this event
Global StudentName, StudentAge, StudentAddress
```

In the code snippet above **three global variables** have been created. These variables **can be used** in **any subroutine** and in **any LiveCode event** so long as they are **declared** at the **start** of the **event** in the same way as shown above.

Parameter Passing

What is a Parameter?

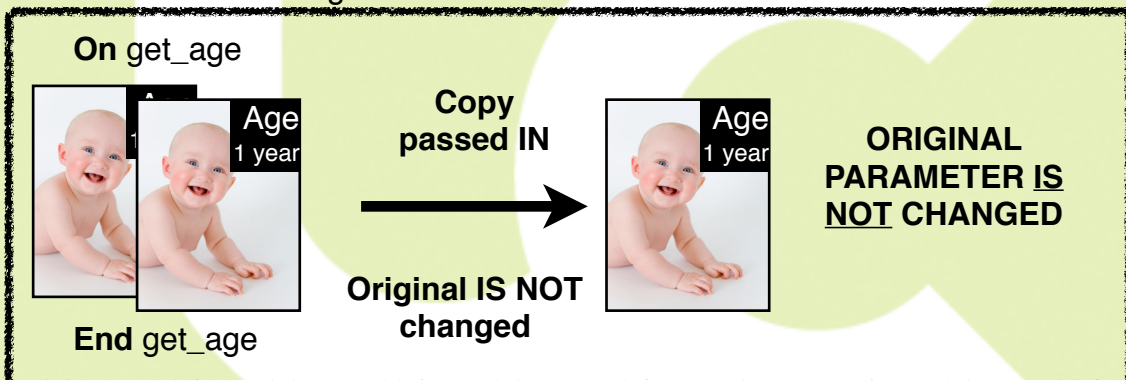
A **parameter** can either be a **variable** or an **array**. When a parameter is used, it can be passed into a sub-routine and **not changed** (passes by value) or passed into a subroutine and **changed** (passed by reference). Only **global variables** and **arrays** can be parameter passed because (as you have already learned), **only** a parameter that is **global** can be used in **more than one subroutine**.

For Higher Computing Science, you need to demonstrate both parameter passing by **value** and by **reference** within the programs you create. It is **vital** you **understand** how it works. Parameter passing works in the **same** way as the **data flow** you do during the **design**.

Parameter Passing by Value

Passing a parameter by **value** is used when a parameter is needed in a subroutine but its value **is not** going to **change** in the **subroutine**.

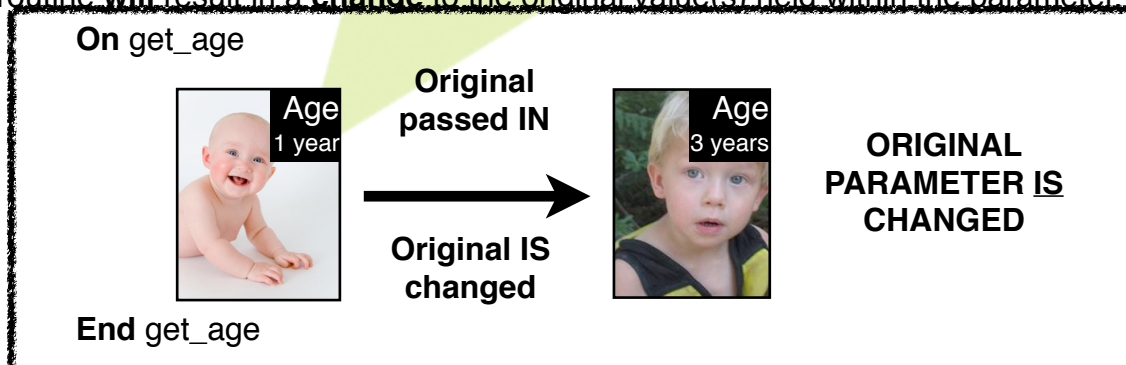
The subroutine will be passed a **copy** of the original parameter, so that the original parameter remains unchanged.



Parameter Passing by Reference

Passing a parameter by **reference** is used when a parameter is needed in a subroutine and its value **is** going to **change** in the **subroutine** when it is passed in.

The subroutine will be passed the original parameter and any changes made in the subroutine **will** result in a **change** to the original value(s) held within the parameter.



Parameter Passing (an example)

PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE

For Higher Computing Science, you need to demonstrate **both** of these methods of **parameter passing** within **all** programs you create. Study the program below carefully. This program calculates the volume of a room and it will be the first program you create. It **includes** parameter passing indicated in **highlighted sections**:

// Here are our global parameters (variables) to be used in this event.

Global RoomLength, RoomBreadth, RoomHeight, RoomVolume

On mouseUp

// After the names of each subroutine, we include all the global parameters used
// within that subroutine separated by a comma.

initialise

get_room_measurements RoomLength, RoomBreadth, RoomHeight

calculate_room_volume RoomLength, RoomBreadth, RoomHeight, RoomVolume

display_room_volume RoomVolume

End mouseUp

On initialise

// The first subroutine does not normally include any parameter passing as this
// involves setting up the parameters to null or 0.

Put 0 into RoomLength

Put 0 into RoomBreadth

Put 0 into RoomHeight

Put 0 into RoomVolume

End initialise

// After the subroutine name below, you will notice that the parameter names have
// an @ symbol before their name. This indicates that the parameters are being
// passed into this subroutine by reference, in other words, they are changing from 0
// (initialised state) to whatever the user enters.

On get_room_measurements @RoomLength, @RoomBreadth, @RoomHeight

Ask "Please enter the length of the room in metres: "

Put it into RoomLength

Ask "Please enter the breadth of the room in metres: "

Put it into RoomBreadth

Ask "Please enter the height of the room in metres: "

Put it into RoomHeight

End get_room_measurements

// After the subroutine name below you will notice that the most of the parameters
// are now being passed by value (no @ sign before the name). This is because the
// values have already been assigned in the previous subroutine and we do not want
// them to change when passed into this subroutine.

//

// The only value which is passed by reference is room_volume as it will be changed
// from its initialised state of 0 to the result of the calculation below.

On calculate_room_volume RoomLength, RoomBreadth, RoomHeight, @RoomVolume

Put (RoomLength * RoomBreadth * RoomHeight) into RoomVolume

End calculate_room_volume

// The only parameter which passed into this subroutine is room_volume. This is
// passed by value as it is the result of the calculation in the previous subroutine and
// we do not want the parameters value to change.

On display_room_volume RoomVolume

Put "The volume of this room is" &&RoomVolume&& "cubic metres." into field "Output"

End display_room_volume

Sequential File Handling

In most programming environments, it is possible to read data in from a text file in a sequential manner, Sequential means one ASCII character after another.

TASK 1: Reading (loading) in a File - About Steve Jobs

To read data from a file into a LiveCode stack we use the following code:

Global MyText

```
On mouseUp
  initialise
  read_file MyText
End mouseUp
```

```
On initialise
  // Initialise the variable to null
  put "" into MyText
End initialise
```

```
On read_file @MyText
  // Ask the user to choose a file to read in
  Answer file "Please choose a file to read into your LiveCode Stack: "

  // If the dialog is not cancelled put the path to the selected file into a variable
  // Use the URL keyword to put the contents of the file into a field
  IF the result is not "cancel" THEN
    Put it into MyText
    Put url ("file:" & MyText) into field "Output"
  END IF
End read_file
```

Let's give it a go:

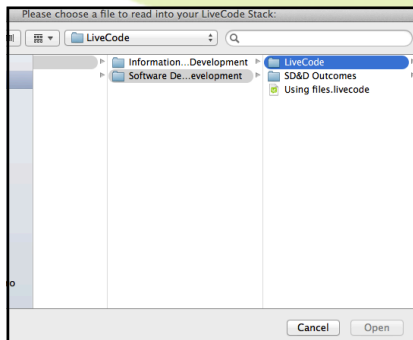
Open the **"About Steve Jobs"** stack. It can be found in:

LiveCode Programming Tasks > 1_Reading In A File.livecode

Using the code above, you are going to read in a text file which contains information about former Apple CEO Steve Jobs. The contents of this file will be read into the main output field.



Key in the code above into the **"Load File"** button.



Once the code has been completed, you will need to test that the program works by reading in the required text file. Run the program, if it works, a new window will appear allowing you to open a text file into the LiveCode stack.

Browse to the file **SampleRead.Txt** and click on "Open", the text from the SampleRead.txt file should be read into the LiveCode stack.



SampleRead.txt

Save

Save the file before you move onto the next task

Save

Sequential File Handling (continued...)

To write data to a file from a LiveCode stack we use the following code:

TASK 2: Writing (saving) to a File - Barclays Premier League



Global OutputContents, MyText

```
On mouseUp
  initialise
  write_file OutputContents, MyText
End mouseUp
```

```
On initialise
  // Initialise the variables to null
  Put "" into OutputContents
  Put "" into MyText
End initialise
```

```
On write_file @OutputContents, @MyText
  // Ask the user to choose where they want to save their file
  Ask file "Please choose a where you want to save the file: "
  IF the result is not "cancel" THEN Put it into MyText

  // Put the contents of the output field into a block variable called OutputContents
  Put field "Output" into OutputContents
  // Put the contents of the variable into the file variable MyText
  Put OutputContents into URL ("file:" & MyText)
End write_file
```

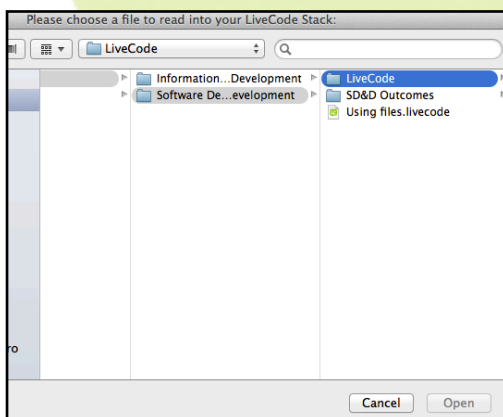
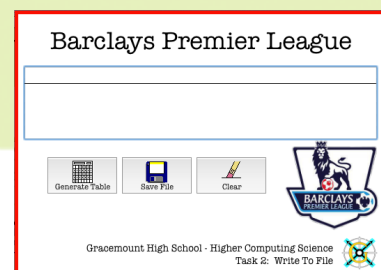
Let's give it a go:

Open the **"Write To File"** stack. It can be found in:

LiveCode Programming Tasks > 2_Write To File.livecode

Using the code above, you are going to write the data that is generated in the output field to a file to be saved to a location of the users choice.

Key in the code above into the **"Save File"** button.



Once the code has been completed, you will need to test that the program works by allowing you to save the contents of the Premier League Table to a file on the hard disk.

Select the Generate Table button and then select the Save File Button. A new window will appear allowing you to save the text file to a location of you choosing. Save it as Premier League Table to your Higher Computing Science Folder.

Once saved, open the text file to check that the contents of the output field has been saved successfully.



Premier League Table

Save

Save the file before you move onto the next task

Save

Algorithms and Sub-Programs (Functions)

PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE

An algorithm is a sequence of steps that are taken to solve a problem. Algorithms can be used within functions, also known as sub programs. A function can be called at any point in a program and as many times as required. A function returns a single value which can be used within a procedure, for example, the largest value held in an array or valid number that is within a set range.

For the Higher Computing Science course, there are **7 standard algorithms** which you need to know about. These are:

- **Count Occurrences:** Returns the number of items found in an array based on the users search.
- **Linear Search:** Returns the search result from a number of items searched.
- **Input Validation:** Returns only a valid number within a set range asked for.
- **Find Maximum:** Returns the highest number from a number of items searched.
- **Find Position of Maximum:** Returns the position in the array where the largest value has been found.
- **Find Minimum:** Returns the lowest number from a number of items searched.
- **Find Position of Minimum:** Returns the position in the array where the smallest value has been found.

Below is the structure of each algorithm used within a function. Notice that at the end of each function, a single value is returned. This value can then be used within any procedure in a program.

Count Occurrences Function

```
Function Count_Occurrences
Repeat with loop = 1 to 20 // Loop for 1 to 20 students
  Ask "Please enter the Computing mark of student number: " & loop
  Put it into arrayMark[loop] // User enters the mark of each student in the array
  IF arrayMark[loop] > 50 THEN
    Add 1 to PassedExam
  END IF
End Repeat
Return PassedExam // Return the number of those that have passed the exam
End Count_Occurrences
```



Linear Search Function

```
Function Linear_Search
// Search for a student name in an array
Ask "Please enter the name of the student: "
Put it into StudentName // User enters the name they wish to search for

Repeat with loop = 1 to 20 // Loop for 1 to 20 students
// If the names array is equal to the search name then..
IF arrayName[loop] = StudentName THEN
  // Put the array data into field "Output1"
  Put arrayName[loop] into StudentName
END IF
End Repeat
Return StudentName // Return the name of the student
End Linear_Search
```



Input Validation

```
Function Validate
// Get the exam mark from the user
Ask "Please enter a whole mark for the exam between 0 and 30: "
Put it into CheckNumber
// Check to make sure the user has entered a valid number within the range expected
Repeat until CheckNumber >= 0 and CheckNumber <= 30 and CheckNumber is an integer
  Ask "Invalid mark, please enter a whole number between 0 and 30: "
  IF the result = "Cancel" THEN exit to top // If the cancel button is pressed, exit to top
  Put it into CheckNumber
End Repeat
Return CheckNumber // Return the validated number that the user has entered
End Validate
```




Algorithms and Sub-Programs (Functions)

PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE


Find Maximum

```
Function Find_Maximum
  Put arrayMark[1] into Maximum // Set Maximum to the first element of array
  Repeat with loop = 1 to 20 // Loop for 1 to 20 students
    IF arrayMark[loop] > Maximum THEN
      Put arrayMark[loop] into Maximum
    END IF
  End Repeat
  Return Maximum // Return the largest number that has been found in the array
End Find_Maximum
```




Find Position of Maximum

```
Function Find_Position_of_Maximum
  Put arrayMark[1] into Maximum // Set Maximum to the first element of array
  Repeat with loop = 1 to 20 // Loop for 1 to 20 students
    IF arrayMark[loop] > Maximum THEN
      Put arrayMark[loop] into Maximum
      Put loop into Position_of_Max
    END IF
  End Repeat
  Return Position_of_Max // Return the position in array that the largest value has been found
End Find_Position_of_Maximum
```




Find Minimum

```
Function Find_Minimum
  Put arrayMark[1] into Minimum // Set Minimum to the first element of array
  Repeat with loop = 1 to 20 // Loop for 1 to 20 students
    IF arrayMark[loop] < Minimum THEN
      Put 1 arrayMark[loop] into Minimum
    END IF
  End Repeat
  Return Minimum // Return the smallest number that has been found in the array
End Find_Minimum
```



Find Position of Minimum

```
Function Find_Position_of_Minimum
  Put arrayMark[1] into Minimum // Set Minimum to the first element of array
  Repeat with loop = 1 to 20 // Loop for 1 to 20 students
    IF arrayMark[loop] < Minimum THEN
      Put arrayMark[loop] into Minimum
      Put loop into Position_of_Min
    END IF
  End Repeat
  Return Position_of_Min // Return the position in array that the smallest value has been found
End Find_Position_of_Minimum
```



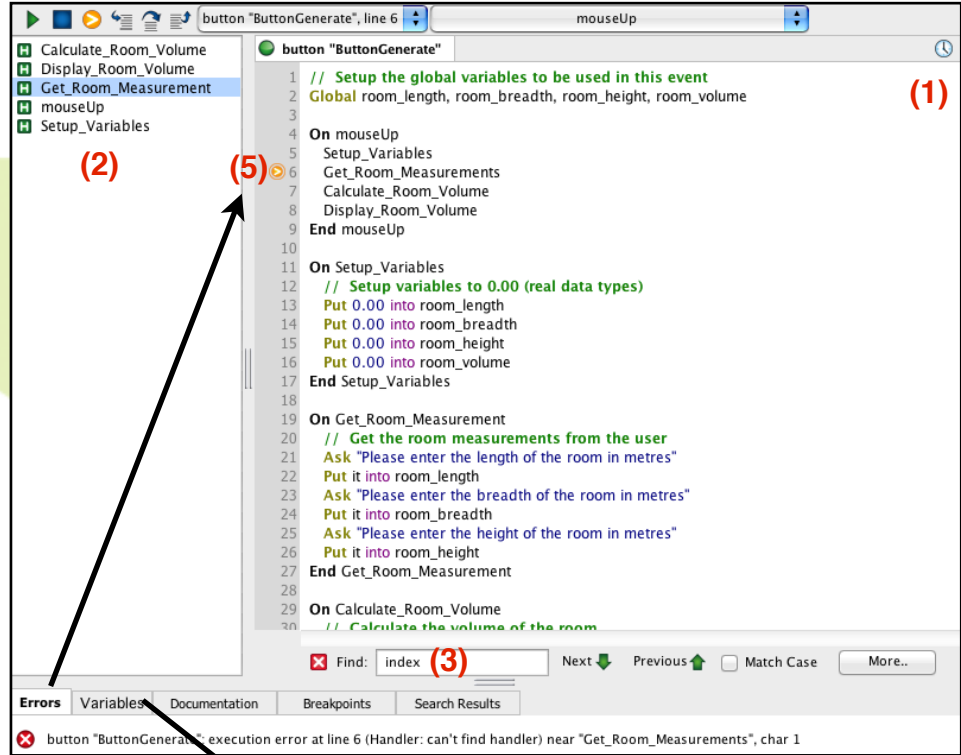
LiveCode Text Editor

PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE

The LiveCode text editor has a wide range of editing features to help make programming easier (1).

The LiveCode text editor gives you a list of all the events you have created within the program (2).

The LiveCode text editor also has the ability to find any term in a program (3).

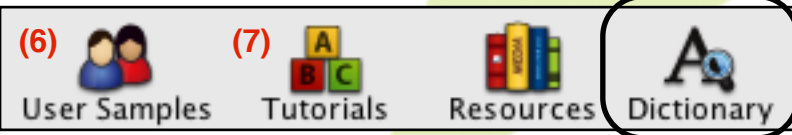


The LiveCode text editor will also give you a complete list of all variables and arrays used in the program (4).

| Errors | Variables | Documentation | Breakpoints | Search Results |
|--------|-----------------|---------------|-------------|----------------|
| | room_breadth | | | 0 |
| | room_height | | | 0 |
| | room_length (4) | | | 0 |
| | room_volume | | | 0 |

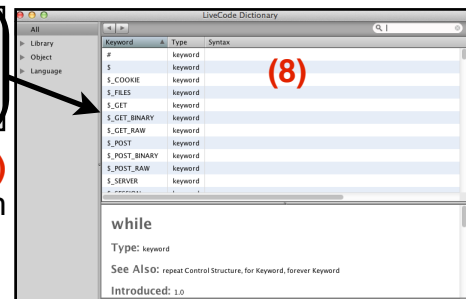
During implementation of a program, LiveCode uses an **interpreter** translator program to help check for errors (see your Software Design and Development notes for a description of this type of translator program).

In the above program, the user is told of an error on line 6 "Get_Room_Measurements". The user needs to look at this line then scan down the program to see that they have in fact called the event on lines 19 and 27 "Get_Room_Measurement" (5).



LiveCode comes with uploaded User Sample (6) programs and Online Tutorials (7) to help them with their programming.

The LiveCode toolbar comes with lots of ways of accessing LiveCode's available commands and resources. The Dictionary, can be called upon to gain access to all of the LiveCode commands (8).



LiveCode Error Checking Tools

PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE

Breakpoints

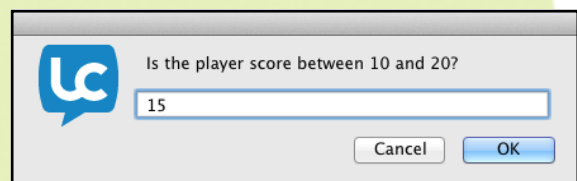
The breakpoint command is used to aid in debugging in your LiveCode program.



It is especially useful for checking to see whether values within variables or arrays are staying in the correct range as the program executes.

When the breakpoint command is used, it will halt the execution of the program on the line and display the current contents of the array or variable as shown below:

```
On mouseUp
  Ask "Is the player score between 10 and 20?"
  Put it into Score
  IF Score >= 10 AND Score <=20 THEN Breakpoint
End mouseUp
```



| Errors | Variables | Documentation | Breakpoints | Search Results |
|--------|-----------|---------------|-------------|----------------|
| | it | | 15 | |
| | score | | 15 | |

Trace Tables

In programming, tracing (Trace Tables) is a specialised use of logging to record information about a program's execution. This information is typically used by programmers for debugging purposes.



| Errors | Variables | Documentation | Breakpoints | Search Results |
|--------|-----------------|---------------|-------------|----------------|
| ▼ | arrayClockSpeed | | | |
| | 1 | 1986 | | |
| | 2 | 550 | | |
| | 3 | 870 | | |
| | 4 | 790 | | |
| | 5 | 1600 | | |
| | 6 | 750 | | |
| ▼ | arrayCost | | | |
| | 1 | 187 | | |
| | 2 | 41 | | |
| | 3 | 150 | ✓ | |
| | 4 | 354 | | |
| | 5 | 575 | | |
| | 6 | 125 | | |
| ▼ | arrayName | | | |
| | 1 | Radeon X2 | | |
| | 2 | GeForce 95 | | |
| | 3 | VaporX | | |
| | 4 | Asus 2 | | |
| | 5 | Nvidia 42X | | |
| | 6 | Voodoo 5 | | |
| ▼ | arrayRam | | | |
| | 1 | 1 | | |
| | 2 | 1 | | |
| | 3 | 2 | | |
| | 4 | 2 | | |
| | 5 | 3 | | |
| | 6 | 1 | | |
| | MaxCards | 6 | | |

Trace tables allow the programmer to see how the contents of a variable or array change as the logic of the program unfolds.

This error detection technique can be useful if the programmer is unsure as to why a value stored within a variable or array is the one which it is.

The current contents of a variable or array can be viewed at any time by selecting the "Variables" tab at the foot of the text editor.

Reminder: What are arrays?

PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE PAUSE

An **array** is a **structured data type** that is used for storing **sets of data within a single variable**.

To put it simply, an array is a **variable** which can store **more than one** piece of data in it so long as it is of the same **data type**.

Like variables, arrays must be **setup** at the start of an event. Look at and understand the example program below. It uses an array called **arrayName** and one variable called **max_students** which sets the **number of student names** to be **stored in the array to 3**.

```
// Setup the global array and variable to be used in this event
Global arrayName, max_students
```

```
On MouseUp
  // Number_of_Students will be set to five so five
  // names will be entered and stored in the array
  Put 3 into max_students
  Get_Student_Name
  Display_Student_Name
End MouseUp
```

```
On Get_Student_Name
  // Start a fixed loop which will repeat five times
  // for each name to be stored in arrayName
  REPEAT with loop = 1 to max_students
    // Get the students name
    Ask "Please enter the name of student: " & loop
    Put it into arrayName[loop]
  END REPEAT
End Get_Student_Name
```

```
On Display_Student_Name
  // Start a fixed loop
  REPEAT with loop = 1 to max_students
    // Put each name entered into arrayName
    // into each line of the Output field using loop
    Put arrayName[loop] into line loop of field "Output"
  END REPEAT
End Display_Student_Name
```

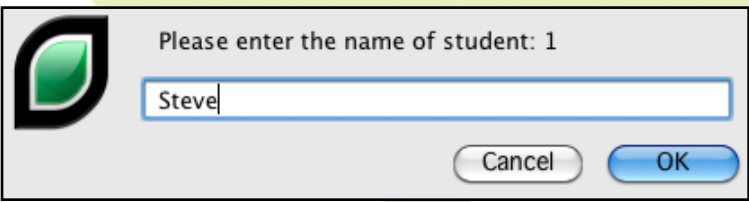
The program knows that **arrayName** is an **array** because of the **[loop]** straight after it.

[loop] indicates the current **position** (space) allocated to the array when it is used in a **loop**. This can be used to store the **users data**. In this program, the loop repeats **three times** as **max_students** is set to **3 in advance**.

So, when the loop **starts**, the user can enter the three names, similar to that below. *Notice that all data in the array are of the same type in this case, string (text):*

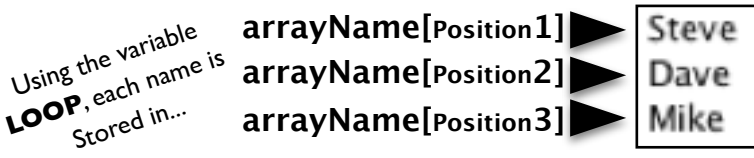
```
REPEAT with loop = 1 to max_students
  arrayName[loop] - "Steve" - put into 1st position of array
  arrayName[loop] - "Dave" - put into 2nd position of array
  arrayName[loop] - "Mike" - put into 3rd position of array
END REPEAT
```

The contents of the array can be displayed using the variable **loop** to ensure all values in each element (space) are displayed in each line (**loop**) 1, 2, 3 of the **Output field**.



- 1st Position of arrayName [loop] stores Steve**
- 2nd Position of arrayName [loop] stores Dave**
- 3rd Position of arrayName [loop] stores Mike**

After the **three** names have been entered, the following Output will be produced in the list **arrayName[loop]** one after the other in the field called **"Output"**:



Using the variable **LOOP**, each name is Stored in...

Think of an Array as being a bit like a multi-level bunk bed. Each bed in the bunk holds one item.

Task 3: String Handling and Concatenation

As well as handling numbers, LiveCode can also perform operations on **text**, also known as **string** variables.

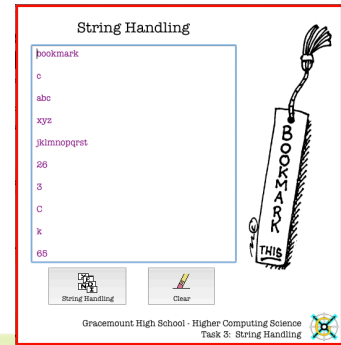
The process of joining **two** or more **strings together** is called **concatenation**. This process is very useful if a program is perhaps required to generate a random **username** or **code** based on a certain number of **characters** contained within a users **forename** and **surname**.

Open the “**String Handling**” stack. It can be found in:

LiveCode Programming Tasks > 3_String Handling.livecode

Work through **each task one** by **one** and **after** completing **each** task, **run** your program to check that your program’s output matches the **expected output**. You **don’t** need to include the **internal commentary** but it’s important that you **understand** how each section of code works as your practical coursework may require you to make use of string handling.

Add the following script to the “**String Handling**” button:



```
On mouseUp
  string_handling
End mouseUp
```

```
On string_handling
```

```
  // Setup the local variables
```

```
  Local FirstWord, SecondWord, CompleteWord, Alphabet
```

```
  //
```

```
  //
```

```
  // -----
```

```
  //
```

```
  //
```

```
  // Task 1
```

```
  // Joining string variables together. This process is called concatenation.
```

```
  Put "book" into FirstWord
```

```
  Put "mark" into SecondWord
```

```
  Put FirstWord into CompleteWord
```

```
  Put SecondWord after CompleteWord
```

```
  Put CompleteWord into line 1 of field "output"
```

```
  // SAVE AND RUN YOUR PROGRAM NOW....
```

```
  // -----
```

```
  //
```

```
  //
```

```
  // Task 2
```

```
  // Create the text to go into the string variable alphabet
```

```
  Put "abcdefghijklmnopqrstuvwxyz" into Alphabet
```

```
  //
```

```
  // -----
```

```
  //
```

```
  //
```

```
  // Task 2 (a)
```

```
  // Put character 3 of the string variable alphabet into line 3 of field output
```

```
  Put char 3 of Alphabet into line 3 of field "Output"
```

```
  // SAVE AND RUN YOUR PROGRAM NOW....
```

```
  //
```

```
  // More tasks over the page.
```



The following output should be produced:

bookmark

The following output should be produced:

c

The String Handling code is continued on the next page

Task 3: String Handling and Concatenation

```

// ----- The following output should be produced:
//
// Task 2 (b) abc
// Put characters 1 to 3 of the string variable alphabet into line 5 of field output
Put char 1 to 3 of Alphabet into line 5 of field "Output"
// SAVE AND RUN YOUR PROGRAM NOW....
// ----- The following output should be produced:
//
// Task 2 (c) xyz
// Put characters 24 to 26 of the string variable alphabet into line 7 of field output
Put char 24 to 26 of Alphabet into line 7 of field "Output"
// SAVE AND RUN YOUR PROGRAM NOW....
// ----- The following output should be produced:
//
// jklmnopqrst
// Task 2 (d)
// Put characters 10 to 20 of the string variable alphabet into line 9 of field output
Put char 10 to 20 of Alphabet into line 9 of field "Output"
// SAVE AND RUN YOUR PROGRAM NOW....
// ----- The following output should be produced:
//
// 26
// Task 2 (e)
// Put the length of the string variable alphabet into line 11 of field output
Put the length of Alphabet into line 11 of field "Output"
// SAVE AND RUN YOUR PROGRAM NOW....
// ----- The following output should be produced:
//
// 3
// Task 2 (f)
// Find the position of a character in a string variable
// This example finds "c" in the string variable alphabet to produce the value of 3
Put offset("c", Alphabet) into line 13 of field "Output"
// SAVE AND RUN YOUR PROGRAM NOW....
// ----- The following output should be produced:
//
// any UPPER case (A-Z)
// Task 3: Number to Character upper case
// Produces a random upper case value from A - Z
// Capital "A" starts at ASCII 65 + 25 other characters of alphabet
put NumToChar (random(26) + 64) into line 15 of field "Output"
// SAVE AND RUN YOUR PROGRAM NOW....
// ----- The following output should be produced:
//
// any lower case (a-z)
// Task 4: Number to Character lower case
// Produces a random lower case value from a - z
// Lower case "a" starts at ASCII 96 + 25 other characters of alphabet
Put NumToChar (random(26) + 95) into line 17 of field "Output"
// SAVE AND RUN YOUR PROGRAM NOW....
// ----- The following output should be produced:
//
// 65
// Task 5: Character to Number
// Produces the ASCII code value for the chosen character
Put CharToNum ("A") into line 19 of field "Output"
// SAVE AND RUN YOUR PROGRAM NOW....
End string_handling

```


Task 4: String Handling and Concatenation - Customer Code Generator

Task

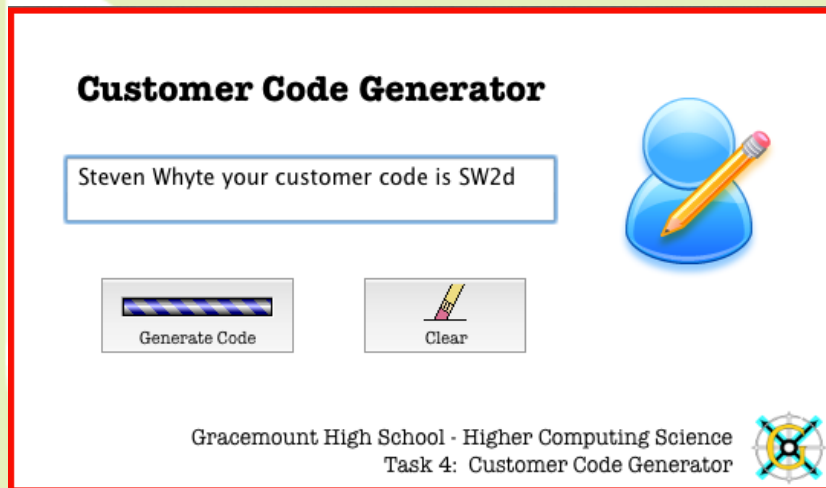
A program is required to generate a customer code. The program should **Ask** the user for their **first name** and **second name**.

Once the program has this information, the customer code should be generated.

The customer code should be made up of:

- The **first character** from both the **first name** and **second name**.
- A **random number** number between **1** and **9**.
- A **random lower case character** (a-z).

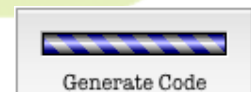
Sample output is shown below assuming the name of **Steven Whyte** has been entered:



Your task is to do the following:

- Produce the program code for this solution using the string handling examples on the previous pages.
- Create a **clear** button and **produce** a simple **script** to **clear** the **output** field.
- **Test** that your solution works by correctly producing the **customer code** as shown above.
- Show the **teacher** your **working** program once **completed**.

Your code should be placed into the “**Generate Code**” button of the “**Customer Code Generator**” stack and the output should be displayed in the **output** field.



The “**Customer Code Generator**” stack can be found in:

LiveCode Programming Tasks > 4_Customer Code Generator.livecode

Good Luck!

Task 5: Flags of the World

Specification

A program is required to ask for the name of a country and display its flag and change the colour of the background to a colour that is contained in the flag.

Each flag has already been placed onto the LiveCode stack and its size has been changed to fit the flag cover graphic. Its size and position has been locked.



Design: Pseudocode for the "Get Flag" button

Stepwise Design *(the main steps of the program - no data flow required)*

1. Choose Country

Stepwise Refinement *(the main step further refined into smaller steps)*

1. Choose Country

1.1 **RECEIVE** GetCountry **FROM** (String) **KEYBOARD**

1.2 **START SWITCH** using GetCountry

1.3 **CASE** "England"

1.4 **SEND** ["Here is the English Flag: "] **TO** line 1 of field "Output"

1.5 **SET** background colour of card **TO** dark pink

1.6 **SET** background colour of graphic FlagCover **TO** dark pink

1.7 **SET** font colour **TO** black

1.8 **SET** the layer of the graphic FlagCover **TO** top

1.9 **SET** the layer of the image England Flag **TO** top

1.10 **CASE** "France"

1.11 **SEND** ["Here is the French Flag: "] **TO** line 1 of field "Output"

1.12 **SET** background colour of card **TO** white

1.13 **SET** background colour of graphic FlagCover **TO** white

1.14 **SET** font colour **TO** black

1.15 **SET** the layer of the graphic FlagCover **TO** top

1.16 **SET** the layer of the image French Flag **TO** top

1.17 **CASE** "Germany"

1.18 **SEND** ["Here is the German Flag: "] **TO** line 1 of field "Output"

1.19 **SET** background colour of card **TO** light yellow

1.20 **SET** background colour of graphic FlagCover **TO** light yellow

1.21 **SET** font colour **TO** black

1.22 **SET** the layer of the graphic FlagCover **TO** top

1.23 **SET** the layer of the image German Flag **TO** top

1.24 **CASE** "Italy"

1.25 **SEND** ["Here is the Italian Flag: "] **TO** line 1 of field "Output"

1.26 **SET** background colour of card **TO** light green

1.27 **SET** background colour of graphic FlagCover **TO** light green

1.28 **SET** font colour **TO** black

1.29 **SET** the layer of the graphic FlagCover **TO** top

1.30 **SET** the layer of the image Italian Flag **TO** top



HAGGIS Design Language

The design is continued on the next page

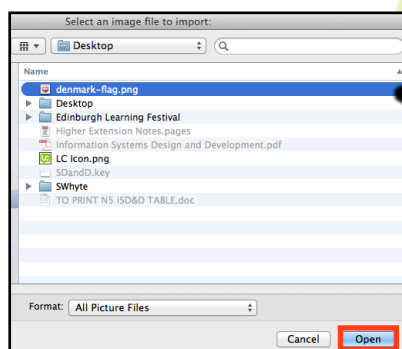
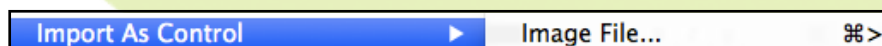
Task 5: Flags of the World

Design (continued)

- 1.31 **CASE** "Scotland"
 1.32 **SEND** ["Here is the Scottish Flag: "] **TO** line 1 of field "Output"
 1.33 **SET** background colour of card **TO** dark blue
 1.34 **SET** background colour of graphic FlagCover **TO** dark blue
 1.35 **SET** font colour **TO** black
 1.36 **SET** the layer of the graphic FlagCover **TO** top
 1.37 **SET** the layer of the image Scottish Flag **TO** top
- 1.38 **CASE** "South Africa"
 1.39 **SEND** ["Here is the South African Flag: "] **TO** line 1 of field "Output"
 1.40 **SET** background colour of card **TO** dark green
 1.41 **SET** background colour of graphic FlagCover **TO** dark green
 1.42 **SET** font colour **TO** black
 1.43 **SET** the layer of the graphic FlagCover **TO** top
 1.44 **SET** the layer of the image South African Flag **TO** top
- 1.45 **CASE** "Spain"
 1.46 **SEND** ["Here is the Spanish Flag: "] **TO** line 1 of field "Output"
 1.47 **SET** background colour of card **TO** light blue
 1.48 **SET** background colour of graphic FlagCover **TO** light blue
 1.49 **SET** font colour **TO** black
 1.50 **SET** the layer of the graphic FlagCover **TO** top
 1.51 **SET** the layer of the image Scottish Flag **TO** top
- 1.52 **CASE** "United States of America"
 1.53 **SEND** ["Here is the USA Flag: "] **TO** line 1 of field "Output"
 1.54 **SET** background colour of card **TO** dark blue
 1.55 **SET** background colour of graphic FlagCover **TO** dark blue
 1.56 **SET** font colour **TO** black
 1.57 **SET** the layer of the graphic FlagCover **TO** top
 1.58 **SET** the layer of the image USA Flag **TO** top
 1.59 **END SWITCH**
- 1.60 **IF** the Output field is empty **THEN**
 1.61 **SET** the background colour **TO** grey **AND** text colour **TO** black
 1.62 **SEND** ["This is country is not listed."] **TO** line 1 of field "Output"
 1.63 **SEND** ["Please try another."] **TO** line 2 of field "Output"
 1.64 **END IF**

Once you have read the above design, implement the program using the code over the page.

Hint: In the extension for this task, you are asked to add some other flags. To import a graphic, select **File** and **Import Graphic as Control** as shown:



Remember to lock the image size and position. You'll find this option in the **graphics properties inspector** (size and position).

Task 5: Using CASE Statements - Flags of the World

Implementation

Place the following code into the Get Flag button. Once complete, run your program to check that it works. Try and add another 3 flags that are similar to the ones that are displayed. You'll find a similar flag by searching on Google images for the name of the country followed by .png (for a transparent background).



```
On mouseUp
  choose_country
End mouseUp
```

```
On choose_country
  // Setup the local variable to be used in this subroutine
  Local GetCountry

  // Setup the card
  Put empty into field "Output"
  Set the BackgroundColor of this card to 220,220,220
  Set the BackgroundColor of graphic "FlagCover" to 220,220,220
  Set the TextColor of field "Output" to 0,0,0
  Set the Layer of graphic "FlagCover" to Top
  Set the Layer of image "Flags.png" to Top
```

```
// Prompt the user for their country
Ask "Please enter the name of the country to display its flag: "
Put it into GetCountry
```

```
// Start a switch statement
SWITCH GetCountry
  // Depending upon what is entered by the user when asked for a
  // country, carry out the following:
```

```
CASE "England"
  Put "Here is the English Flag: " into line 1 of field "Output"
  Set the BackgroundColor of this card to 255,102,102
  Set the BackgroundColor of graphic "FlagCover" to 255,102,102
  Set the TextColor of field "Output" to 0,0,0
  Set the Layer of graphic "FlagCover" to Top
  Set the Layer of image "England.png" to Top
  Break
```

```
CASE "France"
  Put "Here is the French Flag: " into line 1 of field "Output"
  Set the BackgroundColor of this card to 255,255,255
  Set the BackgroundColor of graphic "FlagCover" to 255,255,255
  Set the TextColor of field "Output" to 0,0,0
  Set the Layer of graphic "FlagCover" to Top
  Set the Layer of image "France.png" to Top
  Break
```

```
CASE "Germany"
  Put "Here is the German Flag: " into line 1 of field "Output"
  Set the BackgroundColor of this card to 255,255,51
  Set the BackgroundColor of graphic "FlagCover" to 255,255,51
  Set the TextColor of field "Output" to 0,0,0
  Set the Layer of graphic "FlagCover" to Top
  Set the Layer of image "Germany.png" to Top
  Break
```



The code is continued on the next page

Task 5: Flags of the World

Implementation (continued)

CASE "Italy"

```
Put "Here is the Italian Flag: " into line 1 of field "Output"
Set the BackgroundColor of this card to 102,204,0
Set the BackgroundColor of graphic "FlagCover" to 102,204,0
Set the TextColor of field "Output" to 0,0,0
Set the Layer of graphic "FlagCover" to Top
Set the Layer of image "Italy.png" to Top
Break
```

CASE "Scotland"

```
Put "Here is the Scottish Flag: " into line 1 of field "Output"
Set the BackgroundColor of this card to 51,102,255
Set the BackgroundColor of graphic "FlagCover" to 51,102,255
Set the TextColor of field "Output" to 0,0,0
Set the Layer of graphic "FlagCover" to Top
Set the Layer of image "Scotland.png" to Top
Break
```

CASE "South Africa"

```
Put "Here is the South African Flag: " into line 1 of field "Output"
Set the BackgroundColor of this card to 11,154,25
Set the BackgroundColor of graphic "FlagCover" to 11,154,25
Set the TextColor of field "Output" to 0,0,0
Set the Layer of graphic "FlagCover" to Top
Set the Layer of image "South Africa.png" to Top
Break
```

CASE "Spain"

```
Put "Here is the Spanish Flag: " into line 1 of field "Output"
Set the BackgroundColor of this card to 255,255,0
Set the BackgroundColor of graphic "FlagCover" to 255,255,0
Set the TextColor of field "Output" to 0,0,0
Set the Layer of graphic "FlagCover" to Top
Set the Layer of image "Spain.png" to Top
Break
```

CASE "United States of America"

```
Put "Here is the USA Flag: " into line 1 of field "Output"
Set the BackgroundColor of this card to 128,143,219
Set the BackgroundColor of graphic "FlagCover" to 128,143,219
Set the TextColor of field "Output" to 0,0,0
Set the Layer of graphic "FlagCover" to Top
Set the Layer of image "USA.png" to Top
Break
```

END SWITCH

```
// If the user enters a country not on the list above then set the colour to grey and text to black
// Display the following error message
```

IF field "Output" is empty THEN

```
Set the BackgroundColor of this card to 220,220,220
Set the TextColor of field "Output" to 0,0,0
Put "This country is not listed in the program." into line 1 of field "Output"
Put "Please try another." into line 2 of field "Output"
```

END IF

```
End choose_country
```

To get the background colour and flag cover for the 3 new countries use an RGB colour code by going to the following website:

http://www.rapidtables.com/web/color/RGB_Color.htm



Task 6: Using the Standard Algorithms - Higher Computing Science Marks

Specification

A program is required by a teacher to allow her to store **marks** and **grades** for her Higher Computing Science class. You have been asked to produce a sample program to store the details of **five** students.

This program should allow the teacher to get the **names** and **marks** of the three main topics studied at Higher level. Each mark should be **validated** as **whole numbers** between **0** and **30** (*input validation algorithm*).

Once these details have been keyed in, the program should work out the **percentage mark** and **final grade** based on the student's **percentage** mark. All of these details should then be displayed appropriately in a field called **Output1**.

The program should also allow the teacher to:

- **Search** on a **student name** (*linear search algorithm*).
- **Count** the **occurrences** of each **grade A, B, C, D** and **F** (*count occurrences algorithm*).
- **Find** the **student with the highest percentage** (*find maximum algorithm*).

These details will be displayed in two fields called **Output2** and **Output3**. Sample output from the program is shown below. You may wish to use the **same** test data when it comes to **testing** your program.

This field is called **output1**

Higher Computing Science Marks

Details for Mr Whyte's Higher Computing Science Class

| Student Name | ISD & D Mark | SD & D Mark | Percentage | Grade |
|---------------|--------------|-------------|------------|-------|
| Steven Whyte | 24 | 22 | 77% | A |
| Allan Drain | 30 | 29 | 98% | A |
| Emily Munro | 14 | 13 | 45% | D |
| Jane McGregor | 17 | 18 | 58% | C |
| John Mackie | 5 | 7 | 20% | F |

Count Student Grades

Students who have obtained a Grade A is: 2

Students who have obtained a Grade B is: 0

Students who have obtained a Grade C is: 1

Students who have obtained a Grade D is: 1

The number of students who have obtained a Grade F is: 1

Highest Student Percentage

The student with the highest percentage is with a percentage of 98%.

Get Student Marks

Display All Students

+

Count Grades

↑

Highest Percentage

Clear All Data

Save Student Marks

Find Student

↓

Lowest Percentage

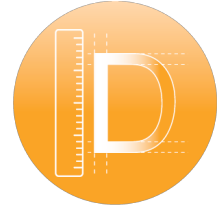
Gracemount High School – Higher Computing Science

Task 6: Higher Computing Science Marks

Read through the **design** of **Get Student Marks** over the page to understand what is involved and then key in the script for this event. The script is supplied for you on pages **28** and **29**.

Task 6: Using the Standard Algorithms - Higher Computing Science Marks

Design for "Get Student Marks" Button



Stepwise Design *(the main steps of the program with data flow)*

1. Setup Variables Arrays Fields
2. Get Student Details In: MaxStudents
In/Out: TeacherName, arrayName, arrayISDD, arraySDD, CheckNumber
3. Calculate Percentage In: MaxStudents, arrayISDD, arraySDD
In/Out: arrayPercentage
4. Get Grade In: MaxStudents, arrayPercentage
In/Out: arrayGrade
5. Display Details In: MaxStudents, arrayName, arrayISDD, arraySDD,
arrayPercentage, arrayGrade
6. Validate In/Out: CheckNumber

Stepwise Refinement *(the main steps further refined into smaller steps)*

1. Setup Variables Arrays Fields

- 1.1 SET MaxStudents TO (Integer) 5
- 1.2 SET CheckNumber TO (Integer) 0
- 1.3 SET TeacherName TO (String) ""
- 1.4 REPEAT with loop = 1 TO MaxStudents
- 1.5 SET arrayName TO (String) ""
- 1.6 SET arrayISDD TO (Integer) 0
- 1.7 SET arraySDD TO (Integer) 0
- 1.8 SET arrayPercentage TO (Integer) 0
- 1.9 SET arrayGrade TO (String) ""
- 1.10 END REPEAT
- 1.11 SEND [Empty] TO field "Output1"
- 1.12 SEND [Empty] TO field "Output2"
- 1.13 SEND [Empty] TO field "Heading1"
- 1.14 SEND [Empty] TO field "Heading2"
- 1.15 SEND [Empty] TO field "Heading3"



HAGGIS Design Language

2. Get Student Details

- 2.1 SEND ["Please enter the name of the teacher who teaches the class: "] TO DISPLAY
- 2.2 RECEIVE TeacherName FROM (String) KEYBOARD
- 2.3 SEND ["Details for " & TeacherName & "'s Higher Computing Science Class"] TO field "Heading1"
- 2.4 REPEAT with loop = 1 TO MaxStudents
- 2.5 SEND ["Please enter the name of student number: "] & loop TO DISPLAY
- 2.6 IF the user selects the cancel button THEN exit to the top of the program
- 2.7 RECEIVE arrayName[loop] FROM (Integer) KEYBOARD
- 2.8 SEND ["Please enter " & arrayName[loop] & "'s mark for Information Systems Design and Development out of 30: "] TO DISPLAY
- 2.9 IF the user selects the cancel button THEN exit to the top of the program
- 2.10 RECEIVE CheckNumber FROM (Integer) KEYBOARD
- 2.11 RECEIVE CheckNumber FROM Validation Function
- 2.12 SEND [CheckNumber] TO arrayISDD[loop]
- 2.13 SEND ["Please enter " & arrayName[loop] & "'s mark for Software Design and Development out of 30: "] TO DISPLAY
- 2.14 IF the user selects the cancel button THEN exit to the top of the program
- 2.15 RECEIVE CheckNumber FROM (Integer) KEYBOARD
- 2.16 RECEIVE CheckNumber FROM Validation Function
- 2.17 SEND [CheckNumber] TO arraySDD[loop]
- 2.18 END REPEAT

The design is continued on the next page

Task 6: Using the Standard Algorithms - Higher Computing Science Marks

Design for “Get Student Marks” Button (continued)

3. Calculate Percentage

- 3.1 REPEAT with loop = 1 TO MaxStudents
- 3.2 SEND [(arrayISDD[loop] + arraySDD[loop]) / 60 * 100] TO arrayPercentage[loop]
- 3.3 END REPEAT

4. Get Grade

- 4.1 REPEAT with loop = 1 TO MaxStudents
- 4.2 IF the arrayPercentage[loop] is greater than 70 THEN SEND [“A”] TO arrayGrade[loop]
- 4.3 IF the arrayPercentage[loop] is between 60 AND 69 THEN SEND [“B”] TO arrayGrade[loop]
- 4.4 IF the arrayPercentage[loop] is between 50 AND 59 THEN SEND [“C”] TO arrayGrade[loop]
- 4.5 IF the arrayPercentage[loop] is between 40 AND 49 THEN SEND [“D”] TO arrayGrade[loop]
- 4.6 IF the arrayPercentage[loop] is less than 40 THEN SEND [“F”] TO arrayGrade[loop]
- 4.7 END REPEAT

5. Display Details

- 5.1 SET NumberFormat TO 0
- 5.2 SEND [“Student Name” & TAB & “ISDD Mark” & TAB & “SDD Mark” & TAB & “Percentage” & TAB & “Grade” TO field “Output1”
- 5.3 REPEAT with loop = 1 TO MaxStudents
- 5.4 SEND [arrayName[loop] & TAB & arrayISDD[loop] & TAB & arraySDD [loop] & TAB & arrayPercentage[loop] & TAB & arrayGrade[loop]] TO field “Output1”
- 5.5 END REPEAT

6. Validate

- 6.1 REPEAT UNTIL CheckNumber is between 0 and 30 and is an integer
- 6.2 SEND [“Invalid mark, enter a whole number between 0 and 30”] TO DISPLAY
- 6.3 IF the user selects the cancel button THEN exit to the top of the program
- 6.4 RECEIVE CheckNumber FROM (Integer) KEYBOARD
- 6.5 END REPEAT
- 6.6 RETURN CheckNumber

Input Validation

Memorise the **structure** of this **algorithm** as you might be asked it in the exam.

Please **READ** the following before you begin the first script.

After carefully reading through the design. You should begin to code the script for the first button called “Get Student Marks”. Key in all of the code over the page **carefully**.

Note. You **do not** need to include **internal commentary** at this point. The commentary is only there to help you understand what is going on.

You will however need to produce internal commentary when it comes to completing your SQA coursework and outcomes.

After completing **each** event, you should **test** that your program is **working correctly** using the supplied **test data**.

Open the “**Higher Computing Science Marks**” stack:

LiveCode Programming Tasks > 6_Higher Computing Science Marks.livecode



Task 6: Using the Standard Algorithms - Higher Computing Science Marks

Implementation: Script "Get Student Marks" Button



Key the following code into the **Get Student Marks** button. You **don't** need to include the **internal commentary**, it is only there to help you understand what is going on.

```
// Setup the global arrays and variables to be used in this event
Global TeacherName, MaxStudents, CheckNumber, arrayName, arrayISDD, arraySDD, arrayPercentage, arrayGrade

// When the mouse up event is detected on this button, execute following subroutines
On mouseUp
  setup_variables_arrays_fields
  get_student_details TeacherName, MaxStudents, CheckNumber, arrayName, arrayISDD, arraySDD
  calculate_percentage MaxStudents, arrayISDD, arraySDD, arrayPercentage
  get_grade MaxStudents, arrayPercentage, arrayGrade
  display_details MaxStudents, arrayName, arrayISDD, arraySDD, arrayPercentage, arrayGrade
End mouseUp

On setup_variables_arrays_fields
  Put 5 into MaxStudents // 5 students maximum
  Put 0 into CheckNumber
  Put "" into TeacherName
  Repeat with loop = 1 to MaxStudents
    Put "" into arrayName[loop]
    Put 0 into arrayISDD[loop]
    Put 0 into arraySDD[loop]
    Put 0 into arrayPercentage[loop]
    Put "" into arrayGrade[loop]
  End Repeat
  Put empty into field "Output1" // Clear the fields
  Put empty into field "Output2"
  Put empty into field "Heading1"
  Put empty into field "Heading2"
  Put empty into field "Heading3"
End setup_variables_arrays_fields

On get_student_details @TeacherName, MaxStudents, @CheckNumber, @arrayName, @arrayISDD, @arraySDD

Ask "Please enter the name of the teacher who teaches the class: "
Put it into TeacherName
Put "Details for " & TeacherName & "'s Higher Computing Science Class" into field "Heading1"

Repeat with loop = 1 to MaxStudents // Loop for 1 to 5 students
  // Get the students name
  Ask "Please enter the name of student number: " & loop
  IF the result = "Cancel" THEN exit to top // If the cancel button is pressed, exit to top
  Put it into arrayName[loop]

  // Get the students validated Information Systems Design and Development mark
  Ask "Enter " & arrayName[loop] & "'s mark for Information Systems Design and Development out of 30: "
  IF the result = "Cancel" THEN exit to top // If the cancel button is pressed, exit to top
  Put it into CheckNumber
  //Run the validation function to make sure a valid ISDD mark has been entered.
  Put Validate (CheckNumber) into arrayISDD[loop] // Put the validated number into the array

  // Get the students validated Software Design and Development mark
  Ask "Enter " & arrayName[loop] & "'s mark for Software Design and Development out of 30: "
  IF the result = "Cancel" THEN exit to top // If the cancel button is pressed, exit to top
  Put it into CheckNumber
  //Run the validation function to make sure a valid SDD mark has been entered.
  Put Validate (CheckNumber) into arraySDD[loop] // Put the validated number into the array
End Repeat
End get_student_details
```

The code is continued on the next page

Task 6: Using the Standard Algorithms - Higher Computing Science Marks

On calculate_percentage MaxStudents, arrayISDD, arraySDD, @arrayPercentage

```
Repeat with loop = 1 to MaxStudents // Loop for 1 to five students
  // Calculate the students percentage mark out of the three tests
  Put (arrayISDD[loop] + arraySDD[loop]) / 60 * 100 into arrayPercentage[loop]
End Repeat
End calculate_percentage
```

On get_grade MaxStudents, arrayPercentage, @arrayGrade

```
// Use of multiple IF's to determine what grade a student gets based on their overall percentage
Repeat with loop = 1 to MaxStudents // Loop for 1 to 5 students
  IF arrayPercentage[loop] >= 70 THEN Put "A" into arrayGrade[loop]
  IF arrayPercentage[loop] >= 60 AND arrayPercentage[loop] <= 69 THEN Put "B" into arrayGrade[loop]
  IF arrayPercentage[loop] >= 50 AND arrayPercentage[loop] <= 59 THEN Put "C" into arrayGrade[loop]
  IF arrayPercentage[loop] >= 40 AND arrayPercentage[loop] <= 49 THEN Put "D" into arrayGrade[loop]
  IF arrayPercentage[loop] < 40 THEN Put "F" into arrayGrade[loop]
End Repeat
End get_grade
```

On display_details MaxStudents, arrayName, arrayISDD, arraySDD, arrayPercentage, arrayGrade

```
Set NumberFormat to "0" // Set the format of any numbers displayed to 0
// Display the headings
Put "Student Name" & tab & "ISD & D Mark" & tab & "SD & D Mark" & tab & "Percentage" & tab & "Grade"
into line 1 of field "Output1" // On the same line.

Repeat with loop = 1 to MaxStudents // Loop for 1 to 5 students
  // Put the array data into field "output1"
  Put arrayName[loop] & tab & arrayISDD[loop] & tab & arraySDD[loop] & tab & arrayPercentage[loop] & "%"
  & tab & arrayGrade[loop] into line loop+2 of field "Output1" // On the same line.
End Repeat

Disable button "GetStudentMarks" // Once all details are displayed, disable this button.
// The button will be enabled again when the user selects clear
```

End display_details

Function Validate

```
// *** Input Validation Function ***
// The validation function will ensure the user has entered a whole number between 0 and 30
Repeat until CheckNumber >= 0 and CheckNumber <= 30 and CheckNumber is an integer
  Ask "Invalid mark, please enter a whole number between 0 and 30."
  IF the result = "Cancel" THEN exit to top // If the cancel button is pressed, exit to top
  Put it into CheckNumber
End Repeat
Return CheckNumber
End Validate
```

Testing

You should now test that your program is working correctly. Key in the name of any teacher followed by the **five names** and **marks** for the **two** assessments below. Check that your **percentage mark** and **grade** is the **same** as below.

If they are the same then you can assume that the **percentage** and **grade** has been **calculated correctly** and your program **works**.

| Student Name | ISD & D Mark | SD & D Mark | Percentage | Grade |
|---------------|--------------|-------------|------------|-------|
| Steven Whyte | 24 | 22 | 77% | A |
| Allan Drain | 30 | 29 | 98% | A |
| Emily Munro | 14 | 13 | 45% | D |
| Jane McGregor | 17 | 18 | 58% | C |
| John Mackie | 5 | 7 | 20% | F |

Task 6: Using the Standard Algorithms - Higher Computing Science Marks

Design for “Count Grades” Button

Stepwise Design *(the main steps of the program with data flow)*

1. Setup Field
2. Count Occurrences In: MaxStudents, arrayGrade



Stepwise Refinement *(the main steps further refined into smaller steps)*

1. Setup Field
 - 1.1 SEND [“Count Grades”] TO field “Heading2”
2. Count Occurrences
 - 2.1 SET AGrade TO (Integer) 0
 - 2.2 SET BGrade TO (Integer) 0
 - 2.3 SET CGrade TO (Integer) 0
 - 2.4 SET DGrade TO (Integer) 0
 - 2.5 SET FGrade TO (Integer) 0
 - 2.6 REPEAT with loop = 1 to MaxStudents
 - 2.7 IF arrayGrade[loop] is equal to “A” THEN Add 1 to AGrade
 - 2.8 IF arrayGrade[loop] is equal to “B” THEN Add 1 to BGrade
 - 2.9 IF arrayGrade[loop] is equal to “C” THEN Add 1 to CGrade
 - 2.10 IF arrayGrade[loop] is equal to “D” THEN Add 1 to DGrade
 - 2.11 IF arrayGrade[loop] is equal to “F” THEN Add 1 to FGrade
 - 2.12 END REPEAT
 - 2.13 SEND [“The number of students who obtained a Grade A is ” & AGrade] TO line 1 of field “Output2”
 - 2.14 SEND [“The number of students who obtained a Grade B is ” & BGrade] TO line 2 of field “Output2”
 - 2.15 SEND [“The number of students who obtained a Grade C is ” & CGrade] TO line 3 of field “Output2”
 - 2.16 SEND [“The number of students who obtained a Grade D is ” & DGrade] TO line 4 of field “Output2”
 - 2.17 SEND [“The number of students who obtained a Grade F is ” & FGrade] TO line 5 of field “Output2”



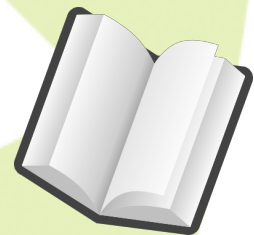
HAGGIS Design Language

Counting Occurrences Algorithm

You must memorise the **structure** of this **algorithm** as you might be asked it in the exam.

Please **READ** the following before you begin the second script.

After carefully reading through the design above. You should begin to code the script for the second button called “**Count Grades**”. Key in all of the code on the next page **carefully** and correct your errors.



This event will count the number of **A, B, C, D** and **F** grades in the sample class of five students using the **arrayGrade**. Each count of grade will then be placed into separate local variables of **AGrade, BGrade, CGrade, DGrade** and **FGrade** as shown above.

After completing the script, you should **test** that your program is working **correctly** by producing the **occurrence** of **each grade obtained**. The predicted test data is shown at the **bottom** of the **next** page.

Task 6: Using the Standard Algorithms - Higher Computing Science Marks

Implementation: Script "Count Grades" Button

Key the following code into the **Count Grades** Button. You **don't** need to include the **internal commentary**, it is only there to help you understand what is going on.



```
// Pass in the global arrays and variables to be used in this event
Global arrayGrade, MaxStudents

// When mouse up event is detected on this button then execute following subroutines
On mouseUp
  setup_field
  count_occurrences MaxStudents, arrayGrade
End mouseUp

On setup_field
  Put "Count Grades" into field "Heading2" // Display the heading "Count Student Grades"
End setup_field

On count_occurrences MaxStudents, arrayGrade
  Local AGrade, BGrade, CGrade, DGrade, FGrade // Setup the local variables

  Put 0 into AGrade // zero the local variables
  Put 0 into BGrade
  Put 0 into CGrade
  Put 0 into DGrade
  Put 0 into FGrade

  // ***Counting Occurrences***
  // Count the number of A, B, C, D and F grades obtained by the class
  Repeat with loop = 1 to MaxStudents // Loop for 1 to 5 students
    IF arrayGrade[loop] = "A" THEN Add 1 to AGrade
    IF arrayGrade[loop] = "B" THEN Add 1 to BGrade
    IF arrayGrade[loop] = "C" THEN Add 1 to CGrade
    IF arrayGrade[loop] = "D" THEN Add 1 to DGrade
    IF arrayGrade[loop] = "F" THEN Add 1 to FGrade
  End Repeat

  // Print the results into field "output2"
  Put "The number of students who have obtained a Grade A is: " & AGrade into line 1 of field "Output2"
  Put "The number of students who have obtained a Grade B is: " & BGrade into line 2 of field "Output2"
  Put "The number of students who have obtained a Grade C is: " & CGrade into line 3 of field "Output2"
  Put "The number of students who have obtained a Grade D is: " & DGrade into line 4 of field "Output2"
  Put "The number of students who have obtained a Grade F is: " & FGrade into line 5 of field "Output2"
End count_occurrences
```

Testing

Test that your program successfully counts the number of each occurrence of grade **A, B, C, D** and **F** and places this into the **output2** field. Your results should look the same as the results below if you are using the **same** test data as you keyed in on page 23.

```
The number of students who have obtained a Grade A is: 2
The number of students who have obtained a Grade B is: 0
The number of students who have obtained a Grade C is: 1
The number of students who have obtained a Grade D is: 1
The number of students who have obtained a Grade F is: 1
```

Task 6: Using the Standard Algorithms - Higher Computing Science Marks

Design for “Find Student” Button



Stepwise Design *(the main steps of the program with data flow)*

1. Setup Fields
2. Find Student
 - In: MaxStudents
 - In: arrayName, arrayISDD, arraySDD, arrayPercentage, arrayGrade

Stepwise Refinement *(the main steps further refined into smaller steps)*

1. Setup Fields
 - 1.1 SEND [Empty] TO field “Output1”
 - 1.2 SEND [“Find Student”] TO field “Heading1”
2. Find Student
 - 2.1 SET StudentName TO (String) “”
 - 2.2 SEND [“Please enter the name of the student: ”] TO DISPLAY
 - 2.3 RECEIVE StudentName FROM (String) KEYBOARD
 - 2.4 SET NumberFormat TO 0
 - 2.5 SEND [“Student Name” & TAB & “ISDD Mark” & TAB & “SDD Mark” & TAB & “Percentage” & TAB & “Grade”] TO line 1 of field “Output1”
 - 2.6 SET found TO false
 - 2.7 REPEAT with loop = 1 to MaxStudents
 - 2.8 IF arrayName[loop] is equal to StudentName THEN
 - 2.9 SEND [arrayName[loop] & TAB & arrayISDD[loop] & TAB & arraySDD [loop] & TAB & arrayPercentage[loop] & TAB & arrayGrade[loop]] TO line 3 of field “Output1”
 - 2.10 SET found TO true
 - 2.11 END IF
 - 2.12 END REPEAT
 - 2.13 IF found is equal to False THEN
 - 2.14 SEND [“No students with that name have been found”] TO line 3 of field “Output1”
 - 2.15 END IF

Linear Search Algorithm

You must memorise the **structure** of this **algorithm** as you might be asked it in the exam.

Please **READ** the following before you begin the third script.

After carefully reading through the design above. You should begin to code the script for the third button called “**Find Student**”. Key in all of the code **carefully**.



This event will **compare** the users **search** with each name in **arrayName** and display the student’s details in the **Output1** field. If no name is found in arrayName then a message explaining that **no students** have been **found** is displayed. Notice that a **boolean** (true/false) variable (Found) is used to **determine** whether or not to display the **no students found** message.

After completing the script, you should **test** that your program is working correctly. The predicted test data is shown at the bottom of the next page assuming the StudentName of “**Steven Whyte**” is entered.

Task 6: Using the Standard Algorithms - Higher Computing Science Marks

Implementation: Script “Find Student” Button

Key the following code into the **Find a Student** button . You **don't** need to include the **internal commentary**, it is only there to help you understand what is going on.



```
// Pass in the global arrays and variables to be used in this event
Global MaxStudents, arrayName, arrayISDD, arraySDD, arrayPercentage, arrayGrade

// When mouse up event is detected on this button then execute following subroutines
On mouseUp
  setup_fields
  find_student arrayName, arrayISDD, arraySDD, arrayPercentage, arrayGrade, MaxStudents
End mouseUp

On setup_fields
  Put empty into field "Output1"
  Put "Find Student" into field "Heading1" // Display heading "Find Student" for this event
End setup_fields

On find_student arrayName, arrayISDD, arraySDD, arrayPercentage, arrayGrade, MaxStudents
  Local StudentName, found // Setup local variables

  Ask "Please enter the name of the student:"
  Put it into StudentName // User enters the name to search

  Set NumberFormat to "0" // Set the format of any numbers displayed to 0
  // Display the headings
  Put "Student Name" & tab & "ISD & D Mark" & tab & "SD & D Mark" & tab & "Percentage" & tab & "Grade"
  into line 1 of field "Output1" // On the same line
  Put false into found // Set the found boolean variable to false

  // ***Linear search***
  // Search for a student based on the name the user has entered
  Repeat with loop = 1 to MaxStudents // Loop for 1 to 5 students
    IF arrayName[loop] = StudentName THEN // If the names array is equal to the search name then..
      // Put the array data into field "Output1"
      Put arrayName[loop] & tab & arrayISDD[loop] & tab & arraySDD[loop] & tab & arrayPercentage[loop]
      & "%" & tab & arrayGrade[loop] into line 3 of field "Output1" // On the same line
      Put true into found // Set found to true
    END IF
  End Repeat

  // If no match found, set found to false and then print a suitable message into field "Output1"
  IF Found = false THEN Put "***** No students with that name have been found
  *****" into line 3 of field "Output1" // On the same line
End find_student
```

Testing

Test that your program successfully **finds** a student once you **search** on their **name**. This should be displayed in the **Output1** field.

| Student Name | ISD & D Mark | SD & D Mark | Percentage | Grade |
|--------------|--------------|-------------|------------|-------|
| Steven Whyte | 24 | 22 | 77% | A |

Task 6: Using the Standard Algorithms - Higher Computing Science Marks

Design for “Highest Percentage” Button



Stepwise Design *(the main steps of the program with data flow)*

1. Setup Field
2. Find Maximum Percentage In: MaxStudents, arrayName, arrayPercentage

Stepwise Refinement *(the main steps further refined into smaller steps)*

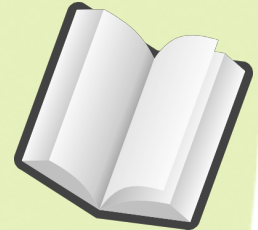
1. Setup Field
 - 1.2 SEND [“Highest Student Percentage ”] TO field “Heading3”
2. Find Maximum Percentage
 - 2.1 SEND [arrayPercentage[1]] TO Maximum
 - 2.2 REPEAT with loop = 1 to MaxStudents
 - 2.3 IF arrayPercentage[loop] is greater than Maximum THEN
 - 2.4 SET arrayPercentage[loop] TO loop
 - 2.5 SET loop TO Position
 - 2.6 END IF
 - 2.7 END REPEAT
 - 2.8 SET NumberFormat TO “0”
 - 2.9 SEND [“The student with the highest percentage is ” & arrayName[Position] “ with a percentage of ” & arrayPercentage[Position] & “%”] TO line 1 of field “Output3”

Find Max Algorithm

You must memorise the **structure** of this **algorithm** as you might be asked it in the exam.

Please **READ** the following before you begin the fourth script.

After carefully reading through the design above. You should begin to code the script for the fourth button called “**Highest Percentage**”. Key in all of the code **carefully**.



This event will **find** and **display** the **highest percentage** mark using **arrayPercentage**. Once found, it will record its position and using the position, can determine the **name** and **percentage mark** of the student with the **highest percentage**.

After completing the script, you should **test** that the program is working correctly. The predicted test data is shown at the **bottom** of the **next page**.

You should also key in the code to **clear** the **output** and **text fields**. This code is displayed at the bottom of the next page and should be assigned to the **clear** button.

Task 6: Using the Standard Algorithms - Higher Computing Science Marks

Implementation: Script “Highest Percentage” Button

Key the following code into the **Highest Percentage** button. You **don't** need to include the **internal commentary**, it is only there to help you understand what is going on. Once completed, **test** that your program correctly identifies the **student** with the **highest percentage mark**. This should be placed into the **output3** field.



```
// Pass in the global arrays and variables to be used in this event
Global MaxStudents, arrayName, arrayPercentage

// When mouse up event is detected on this button then execute following subroutines
On mouseUp
  setup_field
  find_maximum_percentage MaxStudents, arrayName, arrayPercentage
End mouseUp

On setup_field
  Put "Highest Student Percentage" into field "Heading3" // Display heading "Highest Student Percentage" for this event
End setup_field

On find_maximum_percentage MaxStudents, arrayName, arrayPercentage
  Local Maximum, Position // Setup the local variables
  Put 0 into Maximum // Zero the Maximum variable

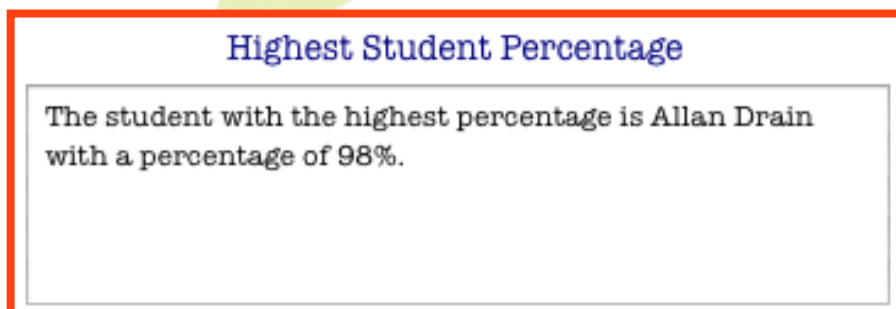
  /**Find Maximum**
  // Find the student with the highest percentage
  Put arrayPercentage[1] into Maximum // Put the first element of arrayPercentage into maximum as a starting point

  Repeat with loop = 1 to MaxStudents // Loop for 1 to 5 students
    IF arrayPercentage[loop] > Maximum THEN // If the percentage array is greater than Maximum
      Put arrayPercentage[loop] into Maximum // Put the value from the array into Maximum
      Put loop into Position // Record the position of the loop
    END IF
  End Repeat

  Set NumberFormat to "0" // Set the format of any numbers displayed to 0
  // Display the position student with the highest percentage in field "Output3"
  Put "The student with the highest percentage is " & arrayName[Position] & " with a percentage of " & arrayPercentage[Position] & "%." into line 1 of field "Output3" // On the same line
End find_maximum_percentage
```

Testing

Test that your program successfully **finds** a student the gained the Highest Percentage. This should be displayed in the **Output3** field.



Task 6: Using the Standard Algorithms - Higher Computing Science Marks

Implementation: Script "Clear All Data"

Once you have tested that your highest percentage button is working, key in the following code for the clear button **carefully**. You don't need to include the internal commentary, it is only there to help you understand what is going on.



// When the mouse up event is detected on this button, execute following actions

On mouseUp

Put empty into field "Output1" // Clear the fields

Put empty into field "Output2"

Put empty into field "Output3"

Put empty into field "Heading1"

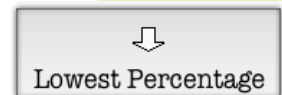
Put empty into field "Heading2"

Put empty into field "Heading3"

Enable button "GetStudentMarks" // Enable the button "Get Students Marks" once the clear button
// has been pressed

End mouseUp

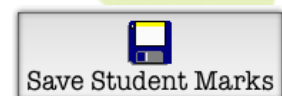
Extension Task 1: Find Lowest Percentage



You are now going to attempt to create the script for **one** button in the program. This button is going to find the **lowest percentage mark** and **display** the **name** and **percentage** mark into **line 1** of the **output3** field.

Remember, if you get stuck, **think!** You've already created the script to find the **highest percentage**. Therefore, the script to find **lowest** percentage is going to be **very similar!** Use the **pseudocode** over the page to help.

Extension Task 2: Save Student Marks



You are now going to attempt to create the script for **one** further button in the program. This button is going to write the contents of what is in the output1 field to a file.

Remember, if you get stuck, **think!** You've already created a script to write to a file in task 2. Use that code to help if you get stuck and also use the **pseudocode** over the page to help.

Extension Task 1 Output

Lowest Student Percentage

The student with the lowest percentage is John Mackie with a percentage of 20%.

Extension Task 2 Output



Results

Test that your Find Minimum and Save Students Marks scripts work correctly. Using your test data from before, the results above and to the right should be produced.

| RESULTS | |
|---|---------------|
| **** HIGHER COMPUTING SCIENCE: STUDENT RESULTS **** | |
| Details for Mr Whyte's Higher Computing Science Class | |
| Student Name: | Steven Whyte |
| ISD & D Mark: | 24 |
| SD & D Mark: | 22 |
| Percentage: | 77% |
| Student Grade: | A |
| Student Name: | Allan Drain |
| ISD & D Mark: | 30 |
| SD & D Mark: | 29 |
| Percentage: | 98% |
| Student Grade: | A |
| Student Name: | Emily Munro |
| ISD & D Mark: | 14 |
| SD & D Mark: | 13 |
| Percentage: | 45% |
| Student Grade: | D |
| Student Name: | Jane McGregor |
| ISD & D Mark: | 17 |
| SD & D Mark: | 18 |
| Percentage: | 58% |
| Student Grade: | C |
| Student Name: | John Mackie |
| ISD & D Mark: | 5 |
| SD & D Mark: | 7 |
| Percentage: | 20% |
| Student Grade: | F |

Task 6: Using the Standard Algorithms - Higher Computing Science Marks

Design for “Lowest Percentage” Button



Stepwise Design (the main steps of the program with data flow)

1. Setup Field
2. Find Maximum Percentage In: MaxStudents, arrayName, arrayPercentage

Stepwise Refinement (the main steps further refined into smaller steps)

1. Setup Field
 - 1.2 SEND [“Lowest Student Percentage ”] TO field “Heading3”
2. Find Maximum Percentage
 - 2.1 SEND [arrayPercentage[1]] TO Minimum
 - 2.2 REPEAT with loop = 1 to MaxStudents
 - 2.3 IF arrayPercentage[loop] is less than Minimum THEN
 - 2.4 SET arrayPercentage[loop] TO loop
 - 2.5 SET loop TO position
 - 2.6 END IF
 - 2.7 END REPEAT
 - 2.8 SET NumberFormat TO “0”
 - 2.9 SEND [“The student with the lowest percentage is ” & arrayName[position] “ with a percentage of ” & arrayPercentage[position] & “%”] TO line 1 of field “Output3”

Find Min Algorithm

You must memorise the structure of this algorithm as you might be asked it in the exam.

Design for “Save Student Marks” Button

Stepwise Design (the main steps of the program with data flow)

1. Initialise
2. Write File In: OutputContents, MyText, MaxStudents, arrayName, arrayISDD, arraySDD, arrayPercentage, arrayGrade

Stepwise Refinement (the main steps further refined into smaller steps)

1. Initialise
 - 1.1 SET OutputContents TO (String) “”
 - 1.2 SET MyText TO (String) “”
2. Write File
 - 2.1 SEND file [“Please choose where you want to save the file: ”] TO DISPLAY
 - 2.2 IF the result is not “cancel” THEN
 - 2.3 RECEIVE MyText FROM (String) source
 - 2.4 END IF
 - 2.5 SEND [“HIGHER COMPUTING SCIENCE: STUDENT RESULTS”] TO OutputContents
 - 2.6 SEND [Return] TO OutputContents
 - 2.7 SEND [“Details for “ & TeacherName & “s Higher Computing Science Class”] TO OutputContents
 - 2.8 SEND [Return] TO OutputContents
 - 2.9 REPEAT with loop = 1 to MaxStudents
 - 2.10 SEND [“Student Name: ” & TAB & arrayName[loop]] TO OutputContents
 - 2.11 SEND [“ISD & D Mark: ” & TAB & arrayISDD[loop]] TO OutputContents
 - 2.12 SEND [“SD & D Mark: ” & TAB & arraySDD[loop]] TO OutputContents
 - 2.13 SET NumberFormat TO “0”
 - 2.14 SEND [“Percentage: ” & TAB & arrayPercentage[loop]] TO OutputContents
 - 2.15 SEND [“Student Grade: ” & TAB & arrayGrade[loop]] TO OutputContents
 - 2.16 SEND [Return] TO OutputContents
 - 2.17 END REPEAT
 - 2.18 CREATE OutputContents FROM url (“File:” & MyText)
 - 2.19 SEND [“File Saved”] TO DISPLAY

Task 7a: Reading, Writing and Linear Search - Country Search

Specification

A program is required to read in a file which contains a list of countries, their capital cities and the population of that capital city as shown below.

Your teacher has provided you with the text document which contains the list. Each item in the list is separated by a full colon “:”

This document must be read into the LiveCode stack and then the user should be able to perform a search on any country in the list in order to produce the correct capital city and population based on the search.

The user should then be able to save their search result which can be opened and viewed in any text editor.

Country Search

| Country | Capital City | Population |
|-------------|--------------|------------|
| China | Beijing | 20693000 |
| Japan | Tokyo | 13189000 |
| Russia | Moscow | 11541000 |
| South Korea | Seoul | 10528774 |
| Indonesia | Jakarta | 10187595 |
| Iran | Tehran | 9110347 |
| Mexico | Mexico City | 885108 |
| Peru | Lima | 8481415 |
| Thailand | Bangkok | 8249117 |
| England | London | 8174100 |

Search Result

Country: Scotland
 Capital City: Edinburgh
 Population: 482640

Gracemount High School - Higher Computing Science
 Task 7a: Country Search

My Search Result

You will find the LiveCode Stack and Countries.txt file in:

LiveCode Programming Tasks > 7_Country Search

Task 7a: Reading, Writing and Linear Search - Country Search

Design: Pseudocode for the “Load Countries File” Button



HAGGIS Design Language

Stepwise Design *(the main steps of the program with data flow)*

1. **Setup Field Variables and Arrays**
2. **Get File** In/Out: MyText
3. **Read File** In: MyText
In/Out: CountryDetails, arrayCountry, arrayCapital, arrayPopulation

Stepwise Refinement *(the main steps further refined into smaller steps)*

1. **Setup Field Variables and Arrays**
 - 1.1 **SET** field “Output1” **TO** Empty
 - 1.2 **SET** MyText **TO** (String) “”
 - 1.3 **SET** CountryDetails **TO** (String) “”
 - 1.4 **SET** arrayCountry **TO** (String) “”
 - 1.5 **SET** arrayCapital **TO** (String) “”
 - 1.6 **SET** arrayCountry **TO** (Integer) 0
2. **Get File**
 - 2.1 **SEND** file [“Please choose a file to read into your LiveCode Stack: ”] **TO DISPLAY**
 - 2.2 **IF** the result is not “cancel” **THEN**
 - 2.3 **RECEIVE** MyText **FROM** (String) source
 - 2.4 **OPEN** url (“File:” & MyText) **FROM** (String) MyText
 - 2.5 **END IF**
3. **Read File**
 - 3.1 **REPEAT** with loop = 1 to the number of lines of MyText
 - 3.2 **SET** line loop of MyText **TO** CountryDetails
 - 3.3 **SPLIT** CountryDetails by colon
 - 3.4 **SET** CountryDetails[1] **TO** arrayCountry[loop]
 - 3.5 **SET** CountryDetails[2] **TO** arrayCapital[loop]
 - 3.6 **SET** CountryDetails[3] **TO** arrayPopulation[loop]
 - 3.7 **SEND** arrayCountry[loop] & TAB & arrayCapital[loop] & TAB & arrayPopulation[loop] **TO** line loop of field “Output1”
 - 3.8 **END REPEAT**

Task 7a: Reading, Writing and Linear Search - Country Search

Implementation

Add the following script to the **Load Countries File** button and **test** that your program allows the user to successfully read in the **Countries.txt** file into the LiveCode Stack.



Global MyText, CountryDetails, arrayCountry, arrayCapital, arrayPopulation

On mouseUp

```
setup_field_variables_and_arrays
get_file MyText
read_file MyText, CountryDetails, arrayCountry, arrayCapital, arrayPopulation
End mouseUp
```

On setup_field_variables_and_arrays

```
Put empty into field "Output1"
Put "" into MyText
Put "" into CountryDetails
Put "" into arrayCountry
Put "" into arrayCapital
Put 0 into arrayPopulation
```

End setup_field_variables_and_arrays

On get_file @MyText

```
// Ask the user to choose a file
Answer file "Please choose a file to read into your LiveCode Stack: "

// If the dialog is not cancelled put the path to the selected file into a variable
// Use the URL keyword to put the contents of the file into a field
IF the result is not "cancel" THEN
Put it into MyText
Put url ("file:" & MyText) into MyText
END IF
```

End get_file

On read_file MyText, @CountryDetails, @arrayCountry, @arrayCapital, @arrayPopulation

```
// Start a fixed loop to place the contents of MyText into CountryDetails
Repeat with loop = 1 to the number of lines of MyText
Put line loop of MyText into CountryDetails
// Split each part of CountryFiles by a colon ":"
Split CountryDetails by colon
// Place each part of country details into 3 separate arrays
Put CountryDetails[1] into arrayCountry[loop]
Put CountryDetails[2] into arrayCapital[loop]
Put CountryDetails[3] into arrayPopulation[loop]

// Display the contents of the file in a field called Output1
Put arrayCountry[loop] & TAB & arrayCapital[loop] & TAB & arrayPopulation[loop] into line loop of
field "Output1" // On the same line
```

End Repeat

End read_file

Testing

Does this part of the program work? Once complete, fix any errors and check that your program reads in the text file of countries into field "Output1" within your LiveCode Stack.

| Country | Capital City | Population |
|-------------|--------------|------------|
| China | Beijing | 20693000 |
| Japan | Tokyo | 13189000 |
| Russia | Moscow | 11541000 |
| South Korea | Seoul | 10528774 |
| Indonesia | Jakarta | 10187595 |
| Iran | Tehran | 9110347 |
| Mexico | Mexico City | 885108 |
| Peru | Lima | 8481415 |
| Thailand | Bangkok | 8249117 |

Task 7a: Reading, Writing and Linear Search - Country Search

Design: Pseudocode for the “Search Country” Button



HAGGIS Design Language

Stepwise Design *(the main steps of the program with data flow)*

1. **Setup Field**
2. **Perform Search** In: MyText, CountryDetails, arrayCountry, arrayCapital, arrayPopulation

Stepwise Refinement *(the main steps further refined into smaller steps)*

1. **Setup Field**
 - 1.1 **SET** field “Output2” **TO** Empty
2. **Perform Search**
 - 2.1 **SET** found **TO** (Boolean) false
 - 2.2 **SET** SearchValue **TO** (String) “”
 - 2.3 **SEND** [“Please enter the country you wish to find the details of: ”] **TO DISPLAY**
 - 2.4 **IF** the result is “cancel” **THEN** exit to top
 - 2.5 **RECEIVE** SearchValue **FROM** (String) **KEYBOARD**
 - 2.6 **REPEAT** with loop = 1 to the number of lines of MyText
 - 2.7 **IF** SearchValue = ArrayCountry[loop] **THEN**
 - 2.8 **SET** loop **TO** position
 - 2.9 **SET** found **TO** true
 - 2.10 **SEND** [“Country: ” & TAB & arrayCountry[Position]] **TO** line 1 of field “Output2”
 - 2.11 **SEND** [“Capital City: ” & TAB & arrayCapital[Position]] **TO** line 2 of field “Output2”
 - 2.12 **SEND** [“Population: ” & TAB & arrayPopulation[Position]] **TO** line 3 of field “Output2”
 - 2.13 **END IF**
 - 2.14 **END REPEAT**
 - 2.15 **IF** found = false **THEN**
 - 2.16 **SEND** [“There are no countries that match your search. Try searching again”] **TO** line 1 of field “Output2”
 - 2.17 **END IF**

Task 7a: Reading, Writing and Linear Search - Country Search

Implementation

Add the following script to the **Search Country** button and **test** that your program allows the user to successfully search for the country Scotland to produce the city of Edinburgh and population 482640.



Global MyText, CountryDetails, ArrayCountry, ArrayCapital, ArrayPopulation

```

On mouseUp
  setup_field
  perform_search MyText, CountryDetails, arrayCountry, arrayCapital, arrayPopulation
End mouseUp

On setup_field
  Put empty into field "Output2"
End setup_field

On perform_search MyText, CountryDetails, arrayCountry, arrayCapital, arrayPopulation
  // Set up the local variables
  Local found, SearchValue, position
  Put 0 into position
  Put false into found

  Ask "Please enter the country you wish to find the details of: "
  IF the result = "Cancel" THEN exit to top
  Put it into SearchValue

  // Start a fixed loop for each of the lines of MyText
  Repeat with loop = 1 to the number of lines of MyText
    // Carry out a linear search searching through each element of ArrayCountry
    IF SearchValue = ArrayCountry[loop] THEN
      // If found, record the position and set the boolean variable to true
      Put loop into position
      Put true into found
      // Put the search result of the output field into a block variable called OutputContents
      Put "Country:" & TAB & arrayCountry[position] into line 1 of field "Output2"
      Put "Capital City:" & TAB & arrayCapital[position] into line 2 of field "Output2"
      Put "Population:" & TAB & arrayPopulation[position] into line 3 of field "Output2"
    END IF
  End Repeat

  // If found is still false after the loop finishes, display a no country found message.
  If found = false THEN
    Put "There are no countries that match your search, try searching again." into line 1 of field "Output2"
  End If

End perform_search

```

Testing

Does this part of the program work? Once complete, fix any errors and carry out a search for the country Scotland.

Check that the details of that country appear in field Output2 as shown on the right.

| Search Result | |
|---------------|-----------|
| Country: | Scotland |
| Capital City: | Edinburgh |
| Population: | 482640 |

Task 7a: Reading, Writing and Linear Search - Country Search

Design: Pseudocode for the “Save Search” Button

Stepwise Design *(the main steps of the program with data flow)*

1. **Initialise**
2. **Write File** in/Out: OutputContents, MyText



HAGGIS Design Language

Stepwise Refinement *(the main steps further refined into smaller steps)*

1. **Initialise**
 - 1.1 **SET** OutputContents **TO** (String) ""
 - 1.2 **SET** MyText **TO** (String) ""
2. **Write File**
 - 2.1 **SEND** file ["Please choose where you want to save the file: "] **TO DISPLAY**
 - 2.2 **IF** the result is not "cancel" **THEN**
 - 2.3 **RECEIVE** MyText **FROM** (String) source
 - 2.4 **END IF**
 - 2.5 **SEND** field "Output2" **TO** OutputContents
 - 2.6 **CREATE** OutputContents **FROM** url ("File:" & MyText)

Task 7a: Reading, Writing and Linear Search - Country Search

Implementation

Add the following script to the **Save Countries File** button and **test** that your program allows the user to successfully save their search result to a file.



Global OutputContents, MyText

```
On mouseUp
  initialise
  write_file OutputContents, MyText
End mouseUp
```

```
On initialise
  // Initialise the variables to null
  Put "" into OutputContents
  Put "" into MyText
End initialise
```

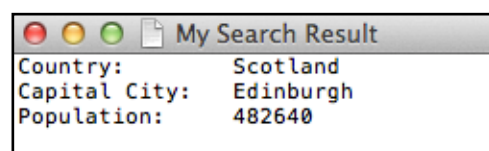
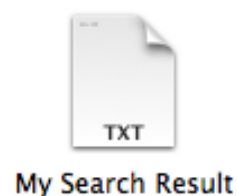
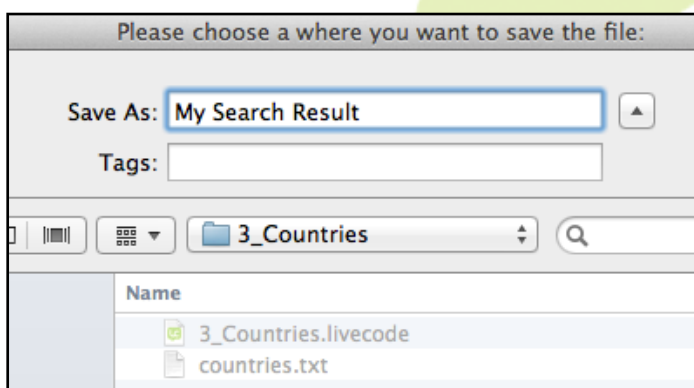
```
On write_file @OutputContents, @MyText
  // Ask the user to choose where they want to save their file
  Ask file "Please choose a where you want to save the file:"
  IF the result is not "cancel" THEN
    Put it into MyText
  END IF
```

```
// Put the search result of the output field into a block variable called OutputContents
Put field "Output2" into OutputContents
```

```
// Put the contents of the variable into the file variable MyText
Put OutputContents into URL ("file:" & MyText)
Answer "File Saved"
End write_file
```

Testing

Does this part of the program work? Once complete, fix any errors and then make sure that you can save the result of your previous search to a suitable folder as shown below.



Task 7b and 7c: Reading, Writing & Standard Algorithms - Country Search

Extension Task



You are going to modify the **Search Country Button** to display and save the following:

Task 7b Country Search (Extension Task 1):

Modify your search so that when you user clicks on the Search Country button, all the capital cities with a population of **between 5000 and 20000 people**.

Hints: In order to complete Extension Task 1, you will need to remove the SearchValue, Position and Found Variables and carry out a complex If Statement using ArrayPopulation. Add a vertical scroll bar to field output2 so you can scroll down the list of countries produced. Your output should be similar to Test Set 1 below. Check that the required data is saved to a file. To get the desired countries saved to a file use the following line in the Save Search button:

Put field "Output2" into OutputContents

Task 7c Country Search (Extension Task 2):

Modify your search so that when the user clicks on the Search Country button, the capital city with the **smallest population is displayed**.

Hints: In order to complete Extension Task 2, you will need to remove the SearchValue, Position and Found Variables and carry out a find position of minimum using ArrayPopulation. Your output should be similar to Test Set 2 below. Check that the required data is saved to a file.

| | |
|---------------|---------------|
| Country: | Mauritius |
| Capital City: | Port Louis |
| Population: | 14725 |
| Country: | Cape Verde |
| Capital City: | Praia |
| Population: | 12546 |
| Country: | South Ossetia |
| Capital City: | Tskhinvali |
| Population: | 15000 |
| Country: | Dominica |
| Capital City: | Roseau |
| Population: | 14847 |
| Country: | Belize |
| Capital City: | Belmopan |
| Population: | 12300 |
| Country: | Aland |
| Capital City: | Mariehamn |
| Population: | 11296 |
| Country: | Grenada |
| Capital City: | St Georges |
| Population: | 7500 |
| Country: | Malta |
| Capital City: | Valletta |
| Population: | 6315 |
| Country: | Liechtenstein |
| Capital City: | Vaduz |
| Population: | 5248 |

TEST SET 1

| | |
|---------------|---------------|
| Country: | Liechtenstein |
| Capital City: | Vaduz |
| Population: | 5248 |

TEST SET 2

Task 8: Working with DataGrids to Display Records - ExtremeTech Graphics



A data grid is just like it sounds. It is a way of **displaying** information from **arrays** and **variables** in a structured **grid** which is clear and easy to read.


A data grid allows the user to quickly **sort** data just by clicking on the **column headings**. This function is **automatically built** into the **datagrid** and does not need to be pre-programmed.

You can also use a data grid to display data **without** setting **tabs**. Instead of using a standard output field, arrays are placed into appropriate columns in the datagrid. These are Setup by the **user** in **advance** using the **properties** of the datagrid.

The data grid that you will be using in this task holds data from **four arrays** which will be held in each row of the dataGrid as a complete record as shown below:

arrayName 

arrayRam 

arrayClockSpeed 

arrayCost 

| Graphics Card Name | Memory (Gb) ▼ | Speed (Mhz) | Cost (£) |
|--------------------|---------------|-------------|----------|
| Nvidia 42X | 3 | 1600 | 575 |
| VaporX | 2 | 870 | 150 |
| Asus 2 | 2 | 790 | 354 |
| Radeon X2 | 1 | 1986 | 187 |
| GeForce 95 | 1 | 550 | 41 |
| Voodoo 5 | 1 | 750 | 125 |

You will see that the records in the data grid above has been sorted by **RAM Memory (GB) field in descending order**. This is done by clicking on the column heading.

Task 8: Working with DataGrids to Display Records - ExtremeTech Graphics

Specification

A program is required by a company called ExtremeTech. The company specialise in the selling of high quality graphics cards.


ExtremeTech require the program in their retail outlets to allow the customer to **browse** through a list of graphics cards and also allow them to **Search on RAM & Cost**.


This will allow a user to search for a graphics card based on their requirements of how much they are willing to spend on a graphics card (**maximum cost**), and how much RAM it must have (**minimum RAM**).


ExtremeTech Graphics Card Database
Displaying All Graphics Cards

| Graphics Card Name | Memory (Gb) | Speed (Mhz) | Cost (£) |
|--------------------|-------------|-------------|----------|
| Nvidia 42X | 3 | 1600 | 575 |
| Asus 2 | 2 | 790 | 354 |
| Radeon X2 | 1 | 1986 | 187 |
| VaporX | 2 | 870 | 150 |
| Voodoo 5 | 1 | 750 | 125 |
| GeForce 95 | 1 | 550 | 41 |

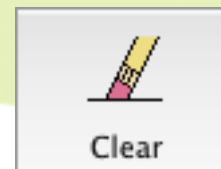
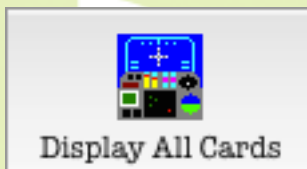
Graphics Cards Found : 6


 Display All Cards


 Search on RAM & Cost


 Clear

Gracemount High School - Higher Computing Science
 Task 8: ExtremeTech Graphics Card Database



What you have to do:

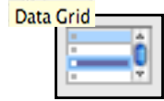
- Open the stack for “**Using Data Grids**”, the location of this can be found below:
 LiveCode Programming Tasks > Extension Task > Using Data Grids.livecode
- Open the script for the “**Display All Cards**” and setup **four global arrays** called:
 - **arrayName, arrayRam, arrayClockSpeed, arrayCost**
- Put the following code **after** you have setup the arrays within “**Display All Cards**” **button** (internal commentary not required but **do read it in order to understand** what is going on):

```
// Setup the Data Grid and the fields
Set the dgData of Group "DataGrid 1" to empty // Clear the data grid
Put empty into field "Total Found" // Clear number of graphic cards found field
Put empty into field "Sub Heading" // Clear the Sub Heading field
```

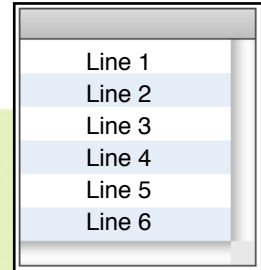
Task 8: Working with DataGrids to Display Records - ExtremeTech Graphics

You are now going to create a **data grid** to go onto the ExtremeTech card. Follow the steps below **carefully**:

Step 1: Select the following icon and **drag** it onto the **centre** of the **card**.



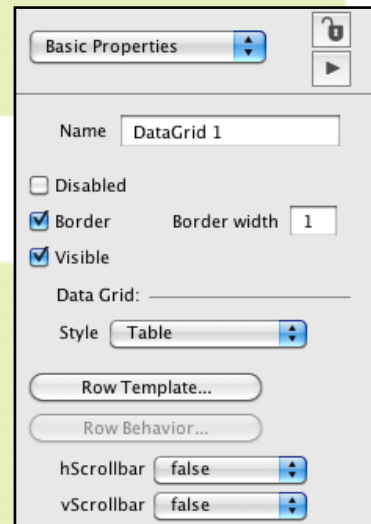
Step 2: **Resize** your data grid to make sure it is **6 lines**. This will fit the data you have placed into your arrays on the previous page.



Step 3: Double-click on the data grid to bring up its **properties**.

Make sure your settings for the data grid match that of what is shown on the right-hand side.

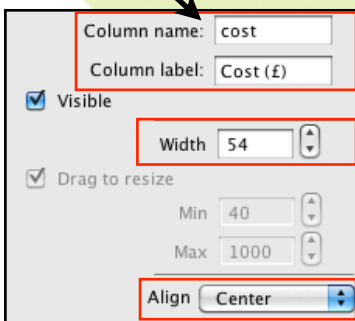
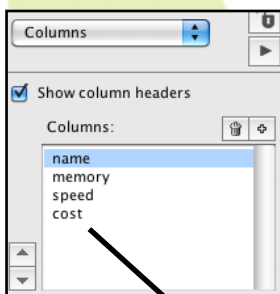
Ensure that the DataGrid name is called **DataGrid 1** and the **hScrollbar** and **vScrollbar** are set to **false** as shown.



Step 4: Select the **Basic Properties** pull down menu and select **Columns**, add **four** columns called:

- **name** with a column label of **Graphics Card Name**
- **memory** with a column label of **Memory (GB)**
- **speed** with a column label of **Speed (MHz)**
- **cost** with a column label of **Cost (£)**

Set the **alignment** of all **columns** to **centre** apart from **name** and adjust the **column width** as appropriate.



ExtremeTech Graphics Card Database

| Graphics Card Name | Memory (Gb) | Speed (Mhz) | Cost (£) |
|--------------------|-------------|-------------|----------|
| | | | |
| | | | |
| | | | |
| | | | |

Graphics Cards Found : 0

Display All Cards

Search on RAM & Cost

Clear

cemount High School – Higher Computing Science
 Task 8: ExtremeTech Graphics Card Database

Step 5: Once complete, your **data grid** and **buttons** should look similar to the screenshot shown on the right.

Task 8: Working with DataGrids to Display Records - ExtremeTech Graphics

Implementation

Key in the following code into the Card Script of the ExtremeTech Stack. This code will put the details of each graphics card into each array.

| Object | Text | Development | View |
|------------------|------|-------------|------|
| Object Inspector | | | |
| Card Inspector | | | |
| Stack Inspector | | | ⌘K |
| Object Script | | | ⌘E |
| Card Script | | | |
| Stack Script | | | |

```

// MAIN CARD
// This script will run when application is first opened. It performs the global assigning
// and initialisation of the arrays containing the graphics cards details.
// Set up arrays as global to allow other buttons to access them.
Global arrayName, arrayRam, arrayClockSpeed, arrayCost, MaxCards

On OpenCard
  initialise MaxCards
  read_graphic_card_data arrayName, arrayRam, arrayClockSpeed, arrayCost
End OpenCard

On initialise @MaxCards
  // Set up the variables, arrays and fields
  Put 6 into MaxCards // Set maximum number of cards, must match number of products in array.
  Repeat with loop = 1 to MaxCards
    Put "" into arrayName[loop]
    Put 0 into arrayRam[loop]
    Put 0 into arrayClockSpeed[loop]
    Put 0 into arrayCost[loop]
  End Repeat
  Set the dgData of Group "DataGrid 1" to empty // Clear the data grid
  Put empty into field "Total Found" // Clear number of graphics cards found field
  Put "Welcome, Select Display All Cards To Begin" into field "Sub Heading" // Display Sub Heading
End initialise

On read_graphic_card_data @arrayName, @arrayRam, @arrayClockSpeed, @arrayCost
  // Set up graphics card details and put each of the details into each element of each array
  Put "Radeon X2" into arrayName[1]
  Put "GeForce 95" into arrayName[2]
  Put "VaporX" into arrayName[3]
  Put "Asus 2" into arrayName[4]
  Put "Nvidia 42X" into arrayName[5]
  Put "Voodoo 5" into arrayName[6]

  Put 1 into arrayRam[1]
  Put 1 into arrayRam[2]
  Put 2 into arrayRam[3]
  Put 2 into arrayRam[4]
  Put 3 into arrayRam[5]
  Put 1 into arrayRam[6]

  Put 1986 into arrayClockSpeed[1]
  Put 550 into arrayClockSpeed[2]
  Put 870 into arrayClockSpeed[3]
  Put 790 into arrayClockSpeed[4]
  Put 1600 into arrayClockSpeed[5]
  Put 750 into arrayClockSpeed[6]

  Put 187 into arrayCost[1]
  Put 41 into arrayCost[2]
  Put 150 into arrayCost[3]
  Put 354 into arrayCost[4]
  Put 575 into arrayCost[5]
  Put 125 into arrayCost[6]
End read_graphic_card_data

```

Task 8: Working with DataGrids to Display Records - ExtremeTech Graphics

Implementation

Key in the following code into the “**Display All Cards**” button of the ExtremeTech Stack.



```
// Allow access to global arrays and variables Setup in main card.
Global arrayName, arrayRam, arrayClockSpeed, arrayCost, MaxCards

On mouseUp
  display_cards arrayName, arrayRam, arrayClockSpeed, arrayCost, MaxCards
End mouseUp

On display_cards arrayName, arrayRam, arrayClockSpeed, arrayCost, MaxCards
  // Setup local variables
  Local theGraphicsCardInfo, CardCount

  // Clear text from the fields
  Put empty into field "Total Found"

  // Display the heading
  Put "Displaying All Graphics Cards" into field "Sub Heading"

  // Zero number of graphics cards found
  Put 0 into CardCount

  // Display all graphics cards
  Repeat with loop = 1 to MaxCards

    // Increment the number of graphics cards displayed.
    // This is also used to determine which records will appear in the data grid.
    Add 1 to CardCount

    // Copy graphics card data from array to the records within the data grid
    Put arrayName[loop] into theGraphicsCardInfo[CardCount]["name"]
    Put arrayRam[loop] into theGraphicsCardInfo[CardCount]["memory"]
    Put arrayClockSpeed[loop] into theGraphicsCardInfo[CardCount]["speed"]
    Put arrayCost[loop] into theGraphicsCardInfo[CardCount]["cost"]
  End Repeat

  Set the dgData of Group "DataGrid 1" to theGraphicsCardInfo // Copy list to data grid
  Put CardCount into field "Total Found" // Display number of graphics cards found
End display_cards
```

Task 8: Working with DataGrids to Display Records - ExtremeTech Graphics

Implementation

Key in the following code into the Search on RAM & Cost button of the ExtremeTech Stack.



```
// Allow access to global arrays and variables Setup in main card.
Global arrayName, arrayRam, arrayClockSpeed, arrayCost, MaxCards

On mouseUp
  search_cards arrayName, arrayRam, arrayClockSpeed, arrayCost, MaxCards
End mouseUp

On search_cards arrayName, arrayRam, arrayClockSpeed, arrayCost, MaxCards
  // Setup local variables
  Local CardCount, theGraphicsCardInfo, MinRam, MaxCost

  // Clear text from the fields
  Put empty into field "Total Found"
  Set the dgData of Group "DataGrid 1" to empty // Clear the data grid

  // Display the heading
  Put "Search Graphics Cards on Minimum RAM and Maximum Cost" into field "Sub Heading"

  Put 0 into CardCount // Zero number of graphics cards found
  Put empty into theGraphicsCardInfo // Clear graphics card list

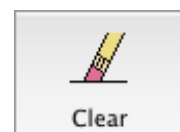
  // Ask the user for the minimum cost and maximum amount of RAM required
  Ask "Please enter the minimum amount of RAM you wish your graphics card to have:"
  Put it into MinRam
  Ask "Please enter the maximum amount you are willing to spend on a new graphics card.:"
  Put it into MaxCost

  // Display graphics cards matching search criteria
  Repeat with loop = 1 to MaxCards
    IF arrayRam[loop] >= MinRam AND arrayCost[loop] <= MaxCost THEN
      Add 1 to CardCount // IF the card matches the search criteria, add card to data grid
      Put arrayName[loop] into theGraphicsCardInfo[CardCount]["name"]
      Put arrayRam[loop] into theGraphicsCardInfo[CardCount]["memory"]
      Put arrayClockSpeed[loop] into theGraphicsCardInfo[CardCount]["speed"]
      Put arrayCost[loop] into theGraphicsCardInfo[CardCount]["cost"]
    END IF
  End Repeat

  Set the dgData of Group "DataGrid 1" to theGraphicsCardInfo // Copy the records to the data grid
  Put CardCount into field "Total Found" // Display number of graphics cards found
End search_cards
```

Implementation

The code for the Clear button is shown below:



```
On mouseUp
  Put empty into field "Sub Heading"
  Set the dgData of Group "DataGrid 1" to empty // Clear the data grid
  Put 0 into field "Total Found"
End mouseUp
```