Plant Growing Simulation

Description of the Program

The program is designed to model how a particular plant in a field would propagate over the course of a variable number of years. The field is divided into a grid of squares, and the contents of each square can be any one of the following:

•	Soil
`S`	A seed
`P`	A plant
`X`	A rock

The field is represented in the program using a two-dimensional character array. The dimensions of the field, and thus the size of the array, are specified in the program's constants `FIELDLENGTH` and `FIELDWIDTH`. The value for `FIELDLENGTH` is set to 20 and `FIELDWIDTH` to 35.

The user is first prompted for a number of years that they wish the program to simulate, which must be an integer between -1 and 5. A number between 0 and 5 indicates the number of years that will be simulated. Entering -1 indicates a desire to step through each year, one at a time, until the user chooses to end the simulation.

The user is then prompted to choose between an empty field as a starting point (which will have a single seed in the middle) or a field loaded from a text file. If the user chooses to load the field from a file, they are prompted for the file name. Each file can only hold one field.

Subsequently, there is no user input, except to advance each year in 'step' mode, and the simulation models the behaviour of plants in the fields 'spring', 'summer', 'autumn' and 'winter' for the number of years specified by the user:

Spring
In spring, all seeds in the field become plants. Subsequently, there is a 1 in 2 chance of a frost. If there is a frost, every third plant is killed off and reverts to soil:

•	•	•	•	•
•	•	•	•	•
•	•	`P`	•	•
•	•	•	•	•
•	•	•	•	•

Summer

In summer, there is a 1 in 3 chance of a drought. In the event of a drought, half of the plants in the simulation (specifically every *other* plant) revert back to soil. If there is no drought, nothing happens during summer:

•	•	•	•	•
•	•	•	•	•
•	•	P	•	•
•	•	•	•	•
•	•	•	•	•

Autumn

In autumn, every square adjacent to a plant is given a seed, unless that adjacent square contains a rock or a plant, in which case it remains a rock or a plant. If multiple seeds are deposited in the same place, only one survives:

•	•	•	•	•
•	S	S	S	•
•	S	P	S	•
•	S	S	S	•
•	•	•	•	•

Winter

In winter, all plants die, reverting to soil, but all seeds remain:

•	•	•	•	•
•	S	S	S	•
•	S	•	S	•
•	S	S	S	•
•	•	•	•	•

After each season, the whole field is displayed in the console, along with the name of the season and the number of the year.

Global Constants

Element	Type	Description
`SOIL`	A character constant	Stores the character to represent soil: **•**
`SEED`	A character constant	Stores the character to represent a seed: **S**
`PLANT`	A character constant	Stores the character to represent a plant: **P**
`ROCKS`	A character constant	Stores the character to represent a rock: **X**
`FIELDLENGTH`	An integer constant	Stores the width of the field, which is also the size of one dimension of the array. Its value in the skeleton program is 20.
`FIELDWIDTH`	An integer constant	Stores the length of the field, which is also the size of the other dimension of the array. Its value in the skeleton program is 35.

Local Variables

Element	Type	Description
`Column`	An integer variable	Declared separately in multiple subroutines, this is used to aid the program in iterating through each row of the field.
`Continuing`	A Boolean variable	Indicates whether another year should run in 'step' mode. Local to `Simulation`.
`Field(,)`	A two-dimensional character array	Each element of this array makes up one square of the field, each of which can be either soil, seed, plant or rock. The array is local, declared and initialised in the `Simulation` subroutines (based on a return value from either `ReadFile` or `InitialiseField`), but it is passed as a parameter to most other subroutines and functions.
`FieldRow`	A string variable	Used to store each line in turn from read from the file specified in `FileName`. Local to `ReadFile`.
`FileName`	A string variable	Entered by the user, the name of a file to load. Local to `ReadFile`.
`Frost`	A Boolean variable	Indicates whether or not there will be frost in the spring. Local to `SimulateSpring`.
`NumberOfPlants`	An integer variable	Used to count the number of plants in the field. Local to `CountPlants`.
`PlantCount`	An integer variable	Used to help model frost and drought in `SimulateSpring` and `SimulateSummer` respectively.
`Rainfall`	An integer variable	Stores an indication of the amount of rain as an integer between 0 and 2, with 0 indicating a drought. Local to `SimulateSummer`.

Element	Type	Description
`Response`	A string variable	Used to store the user's response to the question "do you want to load a file?". Local to `InitialiseField` and `Simulation`.
`Row`	An integer variable	Declared separately in multiple subroutines, this is used to aid the program in iterating through each row of the field.
`Year`	An integer variable	The current year, e.g. 1, 2, 3, etc. Local to `Simulation`.
`Years`	An integer variable	Input by the user, this is the number of years for which the simulation is set to run. Local to `GetHowLongToRun`.
`YearsToRun`	An integer variable	Essentially a copy of `Years` (above), returned from `GetHowLongToRun` to Simulation. `YearsToRun` is the name of the local variable within `Simulation`.

Description of Program Routines

The program functions Ⓕ and procedures Ⓟ are described below.

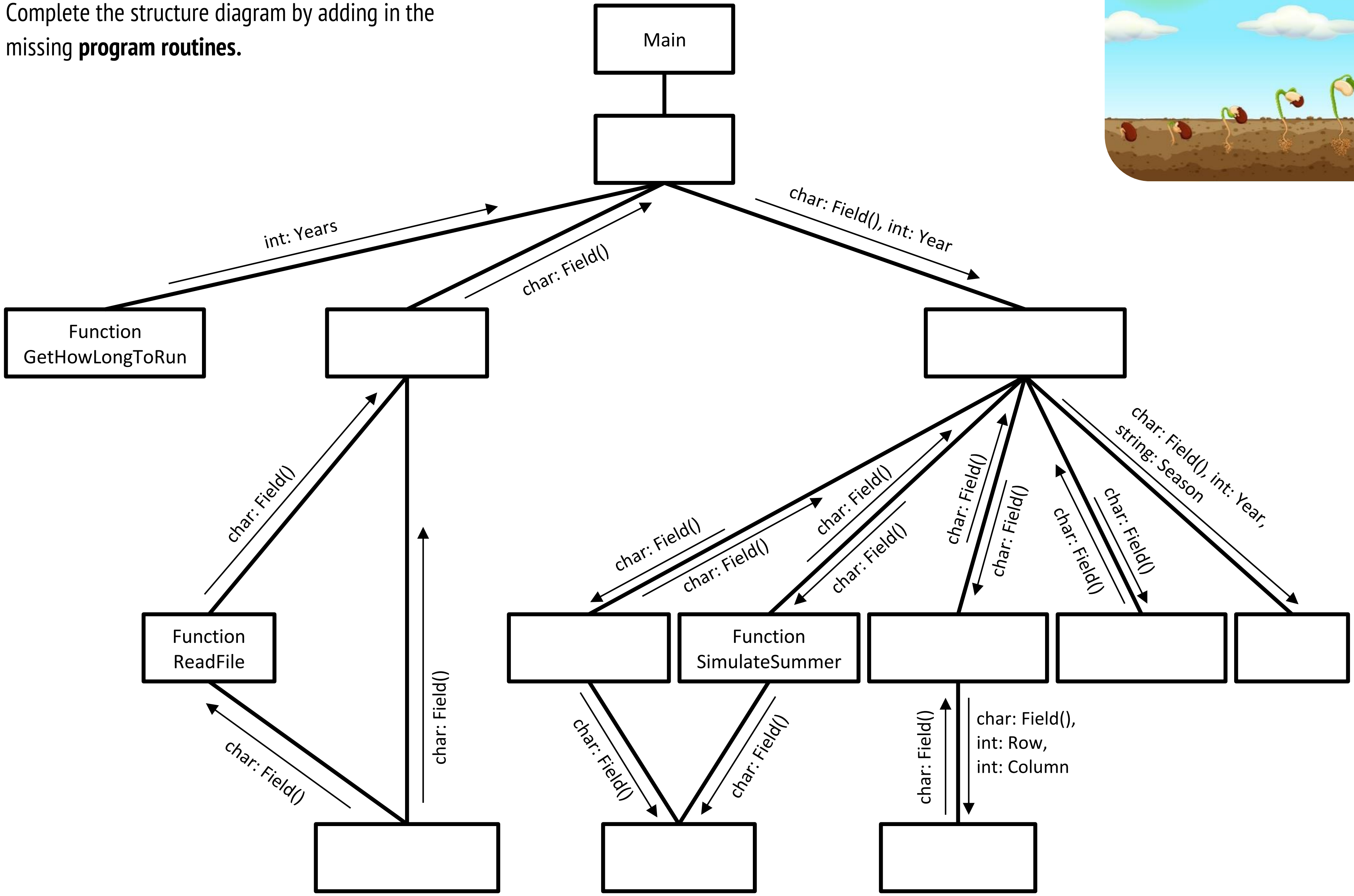
Routine	Description	
`CountPlants` Ⓟ	Receives: `Field` Returns: nothing Called from: `SimulateSpring`, `SimulateSummer`	1. Create variables `Row` and `Column` to allow a nested loop of Field 2. Create variable `NumberOfPlants` to keep a running total, initialised to zero 3. Using a nested loop, iterate through the `Field` array, incrementing `NumberOfPlants` for each plant found 4. Display the number of plants
`CreateNewField` Ⓕ	Receives: nothing Returns: character array Called from: `Readfile`, `InitialiseField`	1. Create variables `Row` and `Column` to allow a nested loop of a new two-dimensional character array called `Field` 2. Initialise Field using the `FIELDLENGTH` and `FIELDWIDTH` constants 3. Using `Row` and `Column`, iterate through all array elements in `Field`, setting each character to represent 'soil' 4. Set the array element at the centre of the `Field` array to represent 'seed' 5. Return the `Field` array to the subroutine that called this routine
`Display` Ⓟ	Receives: `Field`, `Season`, `Year` Returns: nothing Called from: `SimulateOneYear`	1. Create variables `Row` and `Column` to allow a nested loop of Field 2. Initialise `Field` using the `FIELDLENGTH` and `FIELDWIDTH` constants 3. Display the season and the year as a title 4. Using a nested loop, display the contents of `FIELD` as a grid

Routine	Description	
GetHowLongToRun Ⓕ	Receives: nothing Returns: integer Called from: Simulation	1. Declare integer variable Years 2. Display user instructions, prompting the user to enter −1 to enter 'step' mode or 0, 1, 2, 3, 4 or 5 to indicate the number of years to model 3. Prompt user for a response, storing it in Years and returning it to Simulation
InitialiseField Ⓕ	Receives: nothing Returns: character array Called from: Simulation	1. Initialise Field using the FIELDLENGTH and FIELDWIDTH constants 2. Prompt the user as to whether they wish to load a file 3. If 'yes', populate Field with the return value of ReadFile 4. Otherwise, populate Field with the return value of CreateNewField
Main Ⓟ	Receives: nothing Returns: nothing Called from: N/A	1. Initialise random number generator with a new seed (making identical random numbers in consecutive runs extremely unlikely) 2. Call Simulation
ReadFile Ⓕ	Receives: nothing Returns: character array Called from: InitialiseField	1. Create variables Row and Column to allow a nested loop of a new two-dimensional character array called Field 2. Initialise Field using the FIELDLENGTH and FIELDWIDTH constants 3. Prompt the user for a file name and attempt to access that file 4. For each row read from the file, transfer the characters, one at a time, into the Field array 5. Close the file 6. If anything went wrong with either opening the file or transferring data into the Field array, call the CreateNewField routine, which will produce a blank field containing only a seed in the middle 7. Return the Field array, whether populated by the file or the CreateNewField routine, back to the InitialiseField routine
SeedLands Ⓕ	Receives: Field, Row, Column Returns: character array Called from: SimulateAutumn	1. Check that Row and Column variables identify an element within the Field array and not beyond its bounds 2. Check that the element identified by Row and Column variables contains a reference to 'soil' 3. If both (1) and (2) are true, replace 'soil' with 'seed' 4. Return Field to SimulateAutumn
SimulateAutumn Ⓕ	Receives: Field Returns: character array Called from: SimulateOneYear	1. Uses local variables Row and Column, in a nested loop, to iterate through the Field array 2. For each 'plant' element encountered, call SeedLands for each of the eight adjacent elements, i.e. including diagonal adjacency 3. Return Field to SimulateOneYear

Routine	Description
`SimulateOneYear` Ⓟ	Receives: `Field`, `Year` Returns: nothing Called from: `Simulation` 1. Call `SimulateSpring`, then call `Display` 2. Call `SimulateSummer`, then call `Display` 3. Call `SimulateAutumn`, then call `Display` 4. Call `SimulateWinter`, then call `Display` (i.e. simulate each season in turn, displaying the field after each season)
`SimulateSpring` Ⓕ	Receives: `Field` Returns: character array Called from: `SimulateOneYear` 1. Create Boolean variable `Frost` 2. Iterate through the `Field` array, converting all instances of ‘seed’ to ‘plant’ 3. Randomly determine whether there will be frost or not, with a 50% chance of frost 4. If there is frost, iterate through the `Field` array, turning every third instance of ‘plant’ to ‘soil’, and display ‘there has been a frost’ on the screen 5. Call `CountPlants` 6. Return `Field` to `SimulateOneYear`
`SimulateSummer` Ⓕ	Receives: `Field` Returns: character array Called from: `SimulateOneYear` 1. Create Integer variable `Rainfall` 2. By storing a random integer in `Rainfall`, determine whether there will be a drought or not, with a 1 in 3 chance of drought 3. If there is drought, iterate through the `Field` array, turning every other instance of ‘plant’ to ‘soil’, and display ‘there has been a severe drought’ on the screen 4. Call `CountPlants` 5. Return `Field` to `SimulateOneYear`
`SimulateWinter` Ⓕ	Receives: `Field` Returns: character array Called from: `SimulateOneYear` 1. Uses local variables `Row` and `Column`, in a nested loop, to iterate through the `Field` array 2. Replace any instance of ‘plant’ with ‘soil’ (i.e. all plants die) 3. Return `Field` to `SimulateOneYear`
`Simulation` Ⓟ	Receives: nothing Returns: nothing Called from: `Main` 1. Declare `YearsToRun` integer variable and initialise it with a call to `GetHowLongToRun` 2. Declare `Continuing` Boolean, which is set to true for as long as ‘step’ mode continues; if ‘step’ mode is not used, this variable is never used 3. Declare `Response` string, which will accept user input in ‘step’ mode 4. Declare `Field`, the two-dimensional character array, and initialise it using a call to `InitialiseField`, unless a simulation of zero years is requested 5. If ‘step’ mode has *not* been selected, call `SimulateOneYear` the number of times indicated by the user in `GetHowLongToRun`, i.e. the number of full years to be simulated 6. If ‘step’ mode *has* been selected, call `SimulateOneYear` every time the user presses return, indefinitely, until they press ‘x’ then return, at which point the simulation ends 7. Display ‘end of simulation’

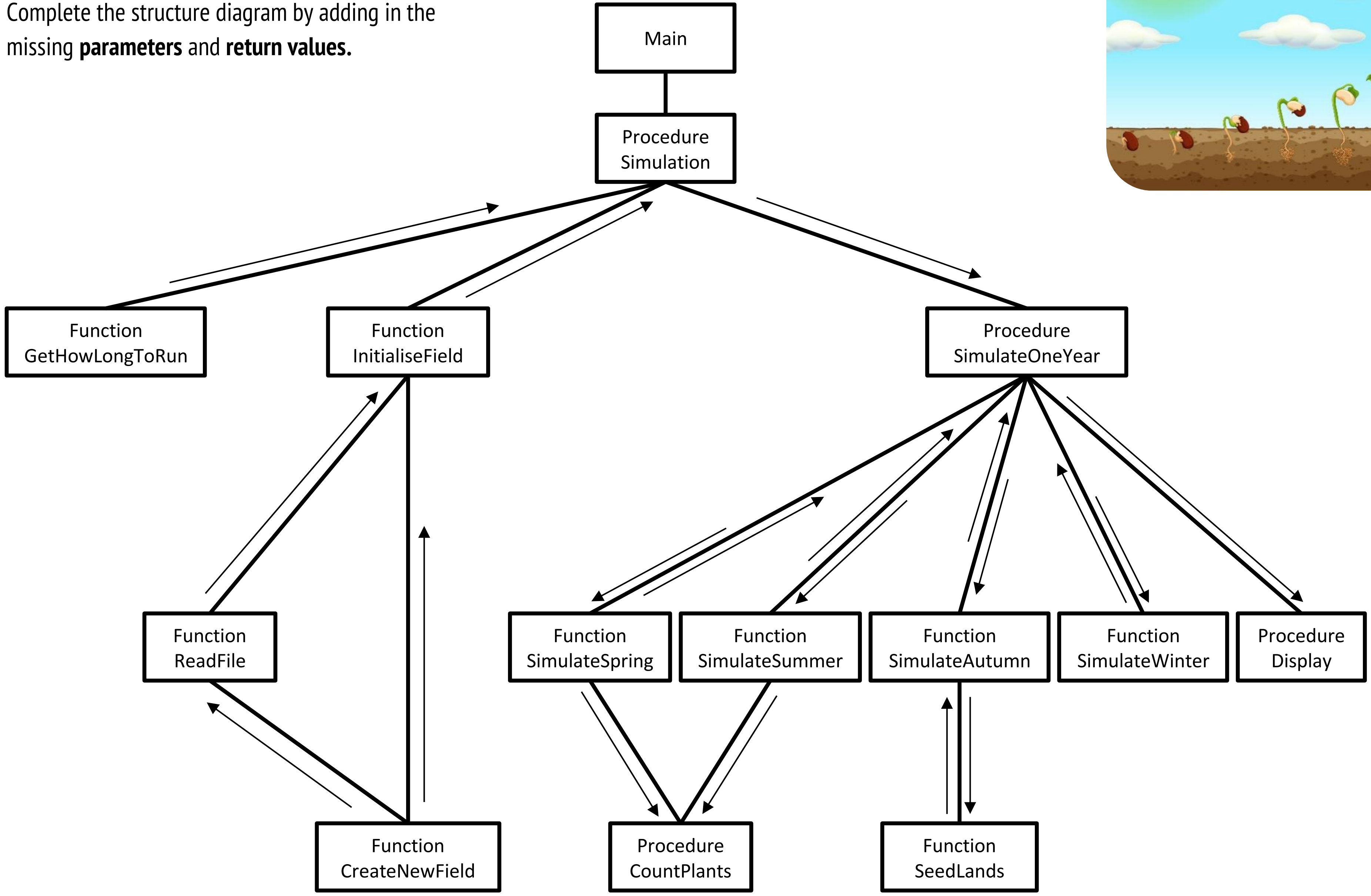
Plant Growing Simulation

Complete the structure diagram by adding in the missing **program routines.**



Plant Growing Simulation

Complete the structure diagram by adding in the missing **parameters** and **return values.**



Main
Procedure Simulation
Function GetHowLongToRun
Function InitialiseField
Procedure SimulateOneYear
Function ReadFile
Function SimulateSpring
Function SimulateSummer
Function SimulateAutumn
Function SimulateWinter
Procedure Display
Function CreateNewField
Procedure CountPlants
Function SeedLands