



Introduction to Programming

Pass Task 5.2: Arrays of Records

Overview

In many programs you will want to store a number of values, where each of those values is a record.

- Purpose:** Learn how to declare and work with arrays of records.
- Task:** Extend your records program (Pass Task 4.1) to use arrays.
- Time:** This task should be completed before the start of week 6.
- Resources:**
- Chapter 6 of the Programming Arcana
 - Swinburne CodeCasts ([YouTube Channel](#), [iTunesU](#))
 - [Using arrays to work with multiple values](#)
 - [Dynamically changing the size of an array](#)
 - Syntax Videos
 - [Pass by Reference \(Var Parameters\)](#), [Pass by Reference \(Const Parameters\)](#), [Pass by Reference \(Out Parameters\)](#), [Arrays](#), [For Loop](#), [Dynamic arrays](#)

Submission Details

You must submit the following files to Doubtfire:

- Code for the program, and a screenshot of it working at the terminal.

Make sure that your task has the following in your submission:

- Code must follow the Pascal coding convention used in the unit (layout, and use of case).
- You are storing and working with multiple values in an array.
- You are using a record and enumeration to store the values.
- The code must compile and the screenshot show it working, and the validations in action.

Instructions

In this task you will extend your program from **Pass Task 4.1** to add an array of the record you have already created. The program can then populate an array of values for the user, output these values, and output some additional summary data.

You will need to make the following changes.

- Create a procedure to **populate** an array of your records from user input via the terminal.
- Create a **print all** procedure to print all of the records from an array.
- Create a procedure to **add** a value to your array.
- Create a function to calculate summary data from your records.
- Adjust **Main** to store a dynamic array of values, read in the number of values the user wants to enter, and set the length of the array, and then show the user a menu. For *example*:

```

procedure: Main
-----
Local Variables:
- addressBook : a dynamic array of ContactData
- numContacts : an Integer for the number of contacts
- option : an Integer for the user's choice from the menu
-----
Steps:
1: Assign numContacts, the result of calling ReadInteger(
    'Enter initial number of contacts: ')
2: Set the Length of addressBook to be numContacts
3: Call Populate Contacts and pass in the addressBook
4: Repeat...
5:     Output a menu '1: Add another contact,
                    2: Print all contacts,
                    3: Find a contact,
                    4: Quit'
5:     Assign option, the result of calling ReadInteger(
        'Enter option: ')
6:     In the case when option is 1:
            Call Add Contact, with reference to addressBook
7:     In the case where option is 2:
            Call Print Contacts and pass in the addressBook
8:     In the case where option is 3:
            Output 'Has contact: ', and the result of
                HasContact( addressBook,
                            ReadString('Which contact name: ') )
9: Until option equals 4
  
```

Notes on the following pages indicate the summary data you should calculate for your program based on the different options you may have chosen in Pass Task 4.1.

Adding a New Value to the Array

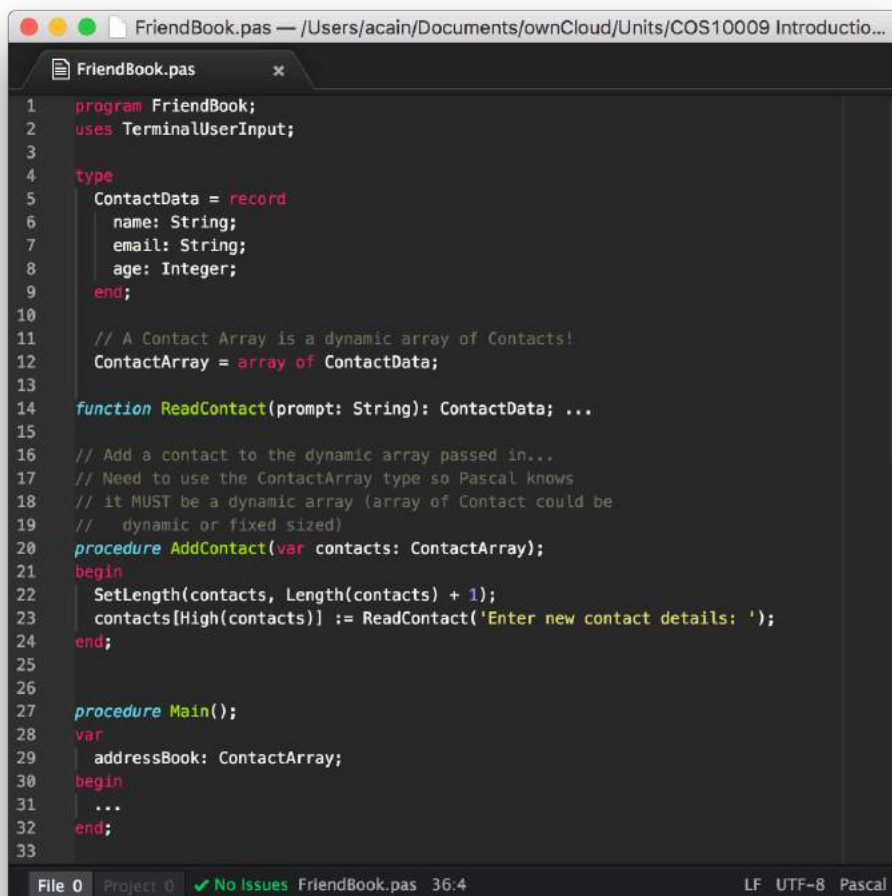
For all program options, you will need to create a procedure to add a new value into your array. The following pseudocode demonstrates how this can be achieved for the Address Book program.

```

procedure: Add Contact
-----
Parameters:
- contacts : a reference to a dynamic array of ContactData
-----
Steps:
1: Call SetLength, of contacts, Length(contacts) + 1
2: Assign the last contact, the value of Read Contact('Enter
new contact details')

```

In Pascal you will need to create your own type to represent the dynamic array. For example, the above pseudocode would need to be coded as follows, with ContactArray being the dynamic array of contacts:



```

1  program FriendBook;
2  uses TerminalUserInput;
3
4  type
5  ContactData = record
6  name: String;
7  email: String;
8  age: Integer;
9  end;
10
11 // A Contact Array is a dynamic array of Contacts!
12 ContactArray = array of ContactData;
13
14 function ReadContact(prompt: String): ContactData; ...
15
16 // Add a contact to the dynamic array passed in...
17 // Need to use the ContactArray type so Pascal knows
18 // it MUST be a dynamic array (array of Contact could be
19 // dynamic or fixed sized)
20 procedure AddContact(var contacts: ContactArray);
21 begin
22   SetLength(contacts, Length(contacts) + 1);
23   contacts[High(contacts)] := ReadContact('Enter new contact details: ');
24 end;
25
26
27 procedure Main();
28 var
29   addressBook: ContactArray;
30 begin
31   ...
32 end;
33

```

File 0 Project 0 ✓ No Issues FriendBook.pas 36:4 LF UTF-8 Pascal

Option 1: Cost Calculator

Allow users to **Add Expense** to the program, and use the following function to calculate the total of all of the expenses of a certain kind.

```
function: Total Expense
-----
Parameters:
- expenses : a constant reference to an array of ExpenseData
- kind : an Expense Kind
Returns (result):
- An Integer with the sum of all the expenses' costs with the
  matching kind
Local Variables:
- i : an Integer
-----
Steps:
1: Assign result, the value 0
2: For each element of expenses (i := 0 to High(expenses))
3:   If the ith expense's kind equals kind
4:     Assign result, the value of result +
       the ith expenses' cost
```

Add a second function named **Largest Expense Of Kind** to find the largest expense of a certain kind.

Create a menu that allows the user to:

1. Add another expense
2. Print all expenses
3. Print total expense for a kind
4. Print largest expense for a kind
5. Quit

The program should loop until the user chooses to quit.

For 3 and 4, the user should be asked to enter the kind to check.

Option 2: Instrument Readings

Allow users to add Readings, and use the following function to calculate the average of all of the readings of a certain kind.

```
function: Average Reading
-----
Parameters:
- readings : a constant reference to an array of SensorData
- kind : a Sensor Kind
Returns (result):
- A Double with the average of all the readings' values with
the matching kind
Local Variables:
- i : an Integer
- numReadings : an Integer
-----
Steps:
1: Assign result, the value 0
2: Assign numReadings, the value 0
3: For each element of readings (i := 0 to High(readings))
4:   If the ith reading's kind equals kind
5:     Assign result, the value of result +
       the ith reading's value
6:     Increase numReadings by 1
7: If numReadings is larger than 0
8:   Assign result the value result / numReadings
```

Add a second function named **Largest Reading Of Kind** to find the largest reading of a certain kind.

Create a menu that allows the user to:

1. Add another reading
2. Print all readings
3. Print average reading for a kind
4. Print largest reading for a kind
5. Quit

The program should loop until the user chooses to quit.

For 3 and 4, the user should be asked to enter the kind to check.

Option 3: High Scores

Use the following function to calculate the highest score for a player.

```
function: Average User Score
-----
Parameters:
- scores : a constant reference to an array of Game Score Data
- name : a String
Returns (result):
- A Integer with the highest score for the player from scores
Local Variables:
- i : an Integer
- numScores: an Integer
-----
Steps:
1: Assign result, the value 0
2: Assign numScores, the value 0
3: For each element of scores (i := 0 to High(scores))
4:     If the ith score's user name equals name
5:         Assign result, the value of result +
           the ith score's value
6:         Increase numScores by 1
7: If numScores is larger than 0
8:     Assign result the value result / numScores
```

Add a second function named **Highest User Score** to find the highest score for a given user name.

Create a menu that allows the user to:

1. Add another score
2. Print all scores
3. Print average user score for a user
4. Print highest score for a user
5. Quit

The program should loop until the user chooses to quit.

For 3 and 4, the user should be asked to enter the name of the user name to check.

Option 4: Bounty Hunter's Hit List

Use the following function to calculate the highest value hit of a certain kind.

```
function: Most Valuable Hit
```

```
-----  
Parameters:
```

- targets : a **constant reference** to an **array** of Target Data
- kind : a Difficulty Kind

```
Returns (result):
```

- A Target Data with the highest value for the given kind

```
Local Variables:
```

- i : an Integer
- ```

```

```
Steps:
```

- 1: **Assign** result, a default value (i.e. set its value to 0, difficulty to the kind parameter, and name to 'No match')
- 2: **For** each other element of targets (i := 0 to High(targets))
- 3:     **If** (the i<sup>th</sup> target's kind) equals kind) **and** ( the i<sup>th</sup> target's value is larger than the result's value )
- 4:         **Assign** result, the value of the i<sup>th</sup> target

Add a second function named **Average Hit Kind Value** to find the average value of hits of a certain kind.

Create a menu that allows the user to:

1. Add another hit
2. Print all hits
3. Print most valuable hit for a kind
4. Print average value of a hit for a kind
5. Quit

The program should loop until the user chooses to quit.

For 3 and 4, the user should be asked to enter the kind to check.

## Option 5: Make your own...

Use the above four declarations guide you to extend your own program. The functions should accept the array and some other value and loop through the array to calculate some summary value (sum, count, average, min, max, etc).

Add **two** functions to calculate values from the data you collected.