Bilkent University

CS 315 Programming Languages Homework #3

SUBPROGRAMS IN DART

Barış Tan Ünal | 22003617

17.12.2022

Table of Contents

1.0 CODE SEGMENT

```
// PART 1: NESTED SUBPROGRAM DEFINITIONS
void main() {

  print( "\nPART 1: NESTED SUBPROGRAM DEFINITIONS" );

  // Outer subprogram.
  double cylinderSurfaceArea( double radius, double height ){

    // Inner subprogram #1 defined inside "cylinderSurfaceArea".
    // It is possible to define nested subprograms in Dart.
    double cylinderLateralArea( double r, double h ){
      return 2 * (3.14159) * r * h;
    }

    // Inner subprogram #2 defined inside "cylinderSurfaceArea".
    double power( double x, int p ){
      if( p == 0 ){
        return 1;
      }
      else{
        return x * power( x, p - 1 );
      }
    }
    return cylinderLateralArea( radius, height ) + 2 * (3.14159) * power( radius, 2 );
  }

  double r = 5;
  double h = 3;
  double a = cylinderSurfaceArea( r, h );
  print( "Surface area of a cylinder with radius $r and height $h is: $a");
  print( "\n------------------------------------------------------------" );


  // PART 2: LOCAL VARIABLE SCOPING
```

```
print( "PART 2: LOCAL VARIABLE SCOPING" );
void foo1( int a ){
  int x = 3;

  void foo2( int b ) {
    int y = 2;
    // All of the variables b, x, and y are visible from here, so no compilation error.
    int i = b * x * y;
    print( "Inside foo2, b * x * y = $i" );
  }
  // Prints 5 * 3 * 2 = 30.
  foo2( 5 );

  // Gives compilation error because variable y is out of scope.
  // return a * y;

  // Does not give a compilation error because both a and x is visible from here.
  int i = a * x;
  print( "In foo1, a * x = $i" );
}

// Prints 4 * 3 = 12.
foo1( 4 );



// PART 3: PARAMETER PASSING METHODS


print( "\n-------------------------------------------------------" );
print( "PART 3: PARAMETER PASSING METHODS" );

// Pass by value.
print( "\tPass by value:" );
int global1 = 5;
void multiplyByThree( int x ){
  x = x * 3;
  print( "After altering passed parameter inside the subprogram, x = $x" );
}
```

```
print( "Before calling multiplyByThree, global1 = $global1" );
multiplyByThree( global1 );
print( "After calling multiplyByThree, global1 = $global1" );
// Global1 variable is printed as 5 after calling multiplyByThree.
// Therefore, we can conclude that default parameter passing method of Dart is pass by value.

// When a list is passed to a subprogram, it acts like pass by reference.
// However, it is still not pass by reference, one can only modify it, but cannot assign a whole different list.
print( "\tPassing lists (acts like pass by reference):" );
void alterList( List<int> evenNumbers ){
  evenNumbers = [ 6, 8, 10 ];
}
void addToList( List<int> evenNumbers ){
  evenNumbers.add( 6 );
}
List<int> evenNumbers = [ 0, 2, 4 ];
print( evenNumbers );
alterList( evenNumbers );
print( evenNumbers );

addToList( evenNumbers );
print( evenNumbers );


// PART 4: KEYWORDS AND DEFAULT PARAMETERS
print( "\n-------------------------------------------------------" );
print( "PART 4: KEYWORDS AND DEFAULT PARAMETERS" );

print( "\tReqiures parameters:" );
void funcRequiredParam( int a1, int a2, int a3){
  print( "In funcRequiredParam, a1 = $a1, a2 = $a2, a3 = $a3" );
}
int i = 1;
int j = 2;
int k = 3;
// We must pass all parameters for reqiured parameters, if we do not program gives a compile time error.
funcRequiredParam( i, j, k );
```

```
print( "\tPositional optional parameters:" );
void funcPositionalParam( int a1, int a2, [int b1 = 5, int b2 = 20]){
  print( "In funcPositionalParam, a1 = $a1, a2 = $a2, b1 = $b1, b2 = $b2" );
}
i = 3;
j = 4;
// We must pass values for requires parameters but positional parameters are optional.
funcPositionalParam( i, j );
// If we want to pass a value for the b2 which is the second positional parameter, we must pass a value for the
previous ones too.
funcPositionalParam( i, j, 10 );
funcPositionalParam( i, j, 10, 100 );

print( "\tNamed optional parameters:" );
void funcDefaultParam( int a1, { int b1 = 5, int b2 = 7 } ){
  print( "In funcDefaultParam, a1 = $a1, b1 = $b1, b2 = $b2" );
}

i = 3;
// We can pass a value for a default parameter if we want.
// Unlike positional parameter passing, we can pass a value that we want to any parameter without passing a value
for previous parameters.
funcDefaultParam( i, b2: 9 );

// But it is also possible not to pass a value for a default parameter.
// In this case its default value of a1 is used.
funcDefaultParam( i );

// PART 5: CLOSURES
print( "\n--------------------------------------------------------" );
print( "PART 5: CLOSURES" );

Function makeMultiplier( int a ){
  return (b) {
    return a * b;
  };
```

```
 }


 // We can create a variable which takes an input as a parameter.
 // Therefore, we can create function-like structures which we can specialize at runtime.


 // Here we create a "multiply by 2" closure.
 var mulBy2 = makeMultiplier(2);
 // Here we create a "multiply by 3" closure.
 var mulBy3 = makeMultiplier(3);


 // It can take parameters like such.
 int x = mulBy2(5);
 int y = mulBy3(10);
 print( "Multiply 5 by 2: $x" );
 print( "Multiply 10 by 3: $y" );


}
```

2.0 EXPLANATION OF EACH DESIGN ISSUE

2.1. NESTED SUBPROGRAM DEFINITIONS

In Dart programming language, it is possible to define nested subprograms. In the code segment, I have defined a function that computes the surface area of a cylinder. In order to compute that, we should first compute the lateral area of the cylinder. For lateral area computation, I have defined another function inside the scope of surface area function which returns the lateral area. Additionally, we also need the area of the circles on both ends of the cylinder. I have defined and implemented another function which recursively computes a power of a number. At the end, I have used both of these inner functions to return the surface area of the cylinder.

Allowing nested subprograms can have some advantages. In my opinion the most beneficial aspect is to organize to the code. If a function is used inside only one particular function, then one can define nested subprograms.

2.2. SCOPE OF LOCAL VARIABLES

Just like other programming languages, lexical scoping is used in Dart. The logic is as follows: When a variable is attempted to be accessed, the innermost scope is searched in the first place. If the program fails to find that variable there, the enclosing scopes are searched from innermost to outermost.

In the example above, there are two nested subprograms foo1 and foo2. The subprogram foo2 can access any variable defined inside itself, in foo1 or those which are global. Therefore, we do not get any compile-time errors. However, when we try to access variable y outside of foo2 inside foo1, we get a compile-time error since it is not visible outside of foo2. That line of code is commented out so that we do not get a compile-time error, but one can uncomment that line and run the program to see that it gives an error.

2.3. PARAMETER PASSING METHODS

Dart uses pass-by-value by default. This means all the parameters that are passed are held in the stack but not the heap. Pass-by-reference is only possible with wrapper functions for primitive types.

In the example above, I have defined a function which multiplies the parameter and prints it. I have defined a global variable which has an initial value of 5. Then I have called the function. We get 15 as output inside the function which is expected. However, after calling the function and printing the global variable, we get 5 as output which was the initial value of the global variable. It shows that the parameter is passed by value and functions cannot alter the primitive-typed parameters that are passed.

However, lists are a bit different. Subprograms can alter the lists by adding or removing elements. But it is not possible to assign an entirely different list to the passed list and see that change outside of the scope of the subprogram. We can conclude that functions that takes lists as parameters acts like it implements pass-by-reference method when we make changes on the original passed list. But the reference that is passed is still passed by value. So, we can conclude that Dart only uses pass-by-value.

2.4. KEYWORD AND DEFAULT PARAMETERS

Dart has three types of parameters: required, positional optional, and named optional. Required parameters are the parameters which are compulsory to be passed in the order that was given in the function definition. If they are not provided when calling the function, program gives a compile-time error. Positional optional parameters have default values therefore it is not compulsory to pass a value for them. However, if a programmer wants to pass a value for a positional parameter, they must pass the previous positional parameters, too. Named optional parameters are very similar to positional optional parameters. The only difference is that it is possible to pass a value for any named parameter without passing values for the previous named parameters. While passing a value for a named parameter, the name of the parameter should be given followed by ':' and the value for it. In the example above, I have given examples for all three cases with proper explanations.

2.5. CLOSURES

Closures are possible to define in Dart since nested subprograms are allowed. Closures are simply modified type of subprograms which returns another subprogram. This way, one can create function-like structures that they can specialize during run-time.

In my example I have defined a closure which creates multipliers. It takes a parameter which specifies the value that it will use to multiply. We can assign the returned value of that closure to a variable. Then, that variable can take a parameter to be multiplied by the previously defined value, which was 2 and 3 respectively. Therefore, we can create infinitely many multiplier-function-like structures with only one definition.

3.0 EVALUATION

It is very easy to implement subprograms in Dart. It is very similar to well-known programming languages such as JavaScript in terms of subprograms, which also allows nested subprograms. By allowing nested subprograms, the readability of the code increases since it make the program look more organized.

The fact that Dart only uses static scoping, increases the readability of the programs. One should only pay attention to enclosing scopes and does not need to track which function called which. However, one cannot take advantage of dynamic scoping. This might decrease the writability since nested scopes are easier to implement with dynamic scoping. However, this does not limit the range of things that can be done, only makes it a bit longer.

Dart only supports pass-by-value. This definitely increases the readability since there is only one possible way to pass a parameter. One does not need to worry about the changes of the value of a primitive type after a function call. However, the absence of pass-by-reference brings a disadvantage when it comes to writability. Being able to pass a parameter with its reference provides a certain level of independence while writing programs in languages such as C++.

There are multiple types of parameters in Dart. This makes Dart more writable because one can force the programmer who uses the functions to pass values for some specific parameters before passing values for other ones with positional optional parameters. Also, it is possible to leave it to the programmer to pass some values for some parameters by giving default values to them. These features increase the writability without decreasing the readability since the representation of default values of parameters and parameter passing methods are very clear.

Although it is a bit difficult to comprehend the concept of closures, they can be very powerful. The ability to define and specialize structures that acts like functions at run-time definitely increases the writability of the language since one can use the same closure many times for purposes that are similar but different. It might be hard to read and understand a closure definition for someone who is not familiar with the concept so we can say that it decreases the readability a little.

Overall, Dart is very resourceful in terms of subprograms with the exception of parameter passing methods where Dart does not support pass-by-reference. I would prefer Dart if I were to implement a program which uses many subprograms.

4.0 LEARNING STRATEGY

First of all, I have looked up for documentation of comments in Dart since I have used comments in all of my code segments to separate sections and explain what my code does clearly. Then I have used the official website of Dart to check the documentation of subprograms. After that, I have read how Dart handles the five design issues given to us. I have checked the information I have read by writing subprograms in the online compiler. I have used print statements all the time to see what happens during run-time. When I had compile-time errors, I have commented the lines with errors and explained why I did so in the code segment. I was not familiar with the concept of closures, therefore I have read the chapter '9.12 Closures' from our textbook to understand the concept.

When I had any unexpected errors that I cannot solve, I have generally checked "www.stackoverflow.com" website to find the similar errors and their solutions. I have used the official website of Dart for further documentation information with examples in some cases. I have used "https://dartpad.dev" as my online compiler, which is the official online compiler of Dart.

I had a little personal communication with my friends in order to understand the problem definition better, especially for the distinction between third and fourth parts. Below, one can access all of the online sources that I have made use of.

5.0 REFERENCES

Sebesta, Robert W. (2016). *Concepts of Programming Languages (11th Edition).* Pearson.

https://dart.dev/guides/language/language-tour#parameters

https://dart.dev/guides/language/language-tour#lexical-scope

https://dart.dev/guides/language/language-tour#lexical-closures

https://subscription.packtpub.com/book/web-development/9781783989560/1/ch01lvl1sec09/functions-and-closures-in-different-scopes

https://toastguyz.com/dart/dart-variable-scope

https://stackoverflow.com/questions/51490066/how-to-return-function-in-dart

https://stackoverflow.com/questions/65643518/dart-pass-by-value-for-int-but-reference-for-list

https://dartpad.dev