Carter Roberts 11/8/2025 Loyola University New Orleans

Assignment: Languages and Paradigms - Subprograms, Parameter Passing, Overloaded Functions, Generic Functions, and Functions as Parameters

Objective:

The purpose of this assignment is to deepen your understanding of subprograms, parameter passing mechanisms, overloaded functions, generic functions, and the use of functions as parameters in programming languages. By completing this assignment, you will gain practical insights into the implementation and application of these concepts.

Instructions:

Part 1: Subprograms (15 points) (read section 9.2 in textbook)

- Define and explain the concept of a subprogram in programming languages.

A subprogram is a function inside of a program, easily re-usable and packaged to be called and returned within the main program.

- Differentiate between procedures and functions. Provide examples to illustrate the distinction (in any language you want).

A procedure does not return a value and instead simply performs whatever action/s it is built to perform; a function, however, returns a value at its termination, which means that its result can be part of mathematical expressions. Here's a Python example of a function, vs. a procedure:

```
1  def functionz(a, b):
2      c = (a + b) / (b - a)
3      return c
4
5  def procedurez():
6      print("What's up homie")
7
8  g = 19
9  print("g =", g)
10 g = functionz(10, 15)
11 print("after function, g =", g)
12 procedurez()
```

```
g = 19
after function, g = 5.0
What's up homie

=== Code Execution Successful ===
```

- Discuss the advantages and disadvantages of using subprograms in software development.

Subprograms are good because they break down large problems into smaller, modular pieces, they're very easily reused, they can be debugged individually, and they're abstract enough that users don't need to understand how they work. The con of subprogram use is that there is a variable level of overhead memory cost depending on the length and complexity of the subprogram as well as whether the parameters are local/dynamic/static or copied. It can really slow things down if you're not cautious.

Part 2: Parameter passing (15 points) (read section 9.5 in textbook)

- Compare and contrast the different parameter passing mechanisms, including pass by value, pass by reference, and pass by name.

Passing by value copies the value of the parameter into the function, but doesn't actually change the original passed parameter. Passing by reference actually places the address of the used variable into the function, which means that it changes the value itself. Passing by name re-evaluates the parameter every time that it's referred to inside the function, which also changes the value itself.

- Provide examples for each parameter passing mechanism to illustrate how they work (in any language you want).

```cpp
#include <iostream>
using namespace std;

void byValue(int a) {
    a++;
    cout << "Value of argument inside byValue func: " << a << endl;
}

void byReference(int &a) {
    a++;
    cout << "Value of argument inside byReference func: " << a << endl;
}

int main() {
    int vale = 10;
    cout << "Value of argument in main before any func: " << vale << endl;
    byValue(vale);
    cout << "Value of argument in main after byValue func: " << vale << endl;
```

```
Value of argument in main before any func: 10
Value of argument inside byValue func: 11
Value of argument in main after byValue func: 10
Value of argument inside byReference func: 11
Value of argument in main after byReference func: 11


=== Code Execution Successful ===
```

I can't find any real languages that I know how to work which use call-by-name. Scala and ALGOL examples I could cite are all too out there for me to explain or properly *get*, which feels like cheating to cite, then.

- Discuss the factors that influence the choice of a parameter passing mechanism in different programming scenarios.

Passing by value is the most optimal when you want a subprogram to not actually override the values inside the variables used, but create a new value using their values. Passing by reference is the most optimal when you want to

update the variable used inside the subprogram. Passing by name is the most optimal when you only want to deal with the variable if it's used in the subprogram; this used to be more valuable because there was far more limited space for computation and if an argument was too costly, they could pass it up.

Part 3: Overloaded subprograms (15 points) (read section 9.9 in textbook)

- Define and explain the concept of function overloading.

If a subprogram is defined with the same name as another subprogram, but it has a different return type or different parameters, it's *overloading* that original function.

- Provide examples of overloaded functions in a programming language of your choice (in any language you want).

```cpp
#include <iostream>
using namespace std;

void blare() {
    cout << "WEE WOO WEE WOO WEE WOO" << endl;
}

void blare(int b) {
    cout << "WEE WOO " << b << " WEE WOO " << b << " WEE WOO " << b << endl;
}

int main() {
    int arg1 = 30;
    blare();
    blare(arg1);

    return 0;
}
```

```
WEE WOO WEE WOO WEE WOO
WEE WOO 30 WEE WOO 30 WEE WOO 30


=== Code Execution Successful ===
```

- Discuss the benefits and potential challenges associated with using overloaded functions.

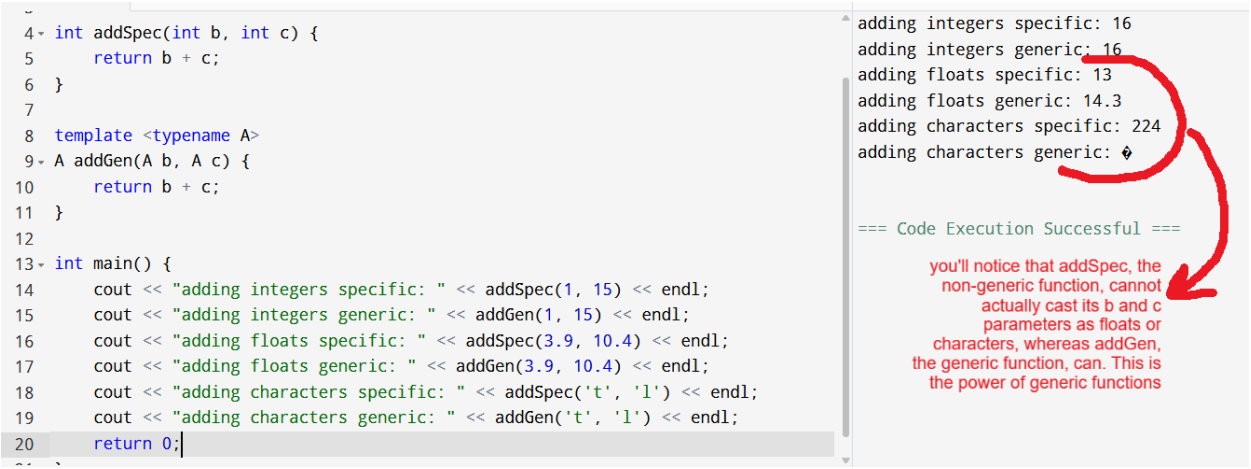
Overloaded functions are good because they make calling functions easier and more efficient since the same "tagline" can be used for multiple different purposes based on the arguments passed. They can be a bit of a problem because it can make reading code kind of a nightmare if there's too many overloaded functions with the same tagline.

Part 4: Generic Functions (15 points) (read section 9.10.1-9.10.4 in your textbook)

- Define and explain the concept of generic functions.

Generic functions are functions which are defined in vague, template-esque terms which are then specified on individual use-case bases depending on how they are called when used.

- Compare and contrast generic functions with regular functions (in any language you want).



- Illustrate the use of generic functions with examples in a programming language that supports generic programming.

See above, as it simultaneously does this.

It's kind of inherent that comparing the use of a generic function to the use of a non-generic function would showcase the use of a generic function, no?

Part 5: Functions as Parameters (15 points) (read sections 9.6 and 9.7 in textbook)

- Discuss the concept of functions as parameters and its significance in programming.

Functions can be passed as parameters in some languages, usually by passing a pointer parameter which matches the *signature* (return type, parameter types & number of them) of the argument function which you intend to pass into the wider function. At least, that's C++. This is utilized to make a semi-generic nested function which will match previous arguments into the arguments of any argument function that has an identical signature to the one passed into the wider function. It's very helpful.

- Provide examples demonstrating the use of functions as parameters (in any language you want).

```cpp
#include <iostream>
using namespace std;

// return type & parameter types/location are called program's "signature"
int add(int x, int y) { return x + y; }
int multiply(int x, int y) { return x * y; }
// parameterized subprogram has to match signature of used function
int invoke(int x, int y, int (*func)(int, int)) {
    return func(x, y);
}

int main() {
    // use & to match parameter's *
    cout << invoke(20, 10, &add) << endl;
    cout << invoke(20, 10, &multiply) << endl;
    return 0;
}
```

```
30
200


=== Code Execution Successful ===
```

- Explore how functions as parameters contribute to the development of flexible and modular code.

Functions as parameters further enhance the reusability of functions in a manner already explained in the first question of this part. They further abstract the code and clean up parts of code that are similar and can be reduced down to different uses of the same nested function.

Part 6: Coroutine: (10 points) (read section 9.13 in your textbook)

- Discuss the concept of coroutines and its significance in programming.

Coroutines are functions which allow for multiple points of entry and exit, allowing for them to be effectively paused and resumed. This allows for an enhanced flow of control and reduces overhead for continuous tasks massively.

- Provide examples demonstrating the use of coroutines in any language; use internet to find the answer (recommend C# or Python).

Marangon, J.D (2024) developed a Python coroutine example which utilizes yield, send(), next() & close() to simulate a calculator which continually runs averages by yielding in an endless loop, using next() to bring up to bat the first send(), and send() repeatedly to share new values to add towards new averages, before close()-ing the whole thing:

```python
def average():
    total = count = average = 0

    while True:
        number = yield average  # Yield the current average and wait for a new number to be sent
        if not isinstance(number, int):
            break  # Exit if not sent and integer value, ending the coroutine
        total += number
        count += 1
        average = total / count  # Calculate new average


co = average() # Instantiate the coroutine

next(co)  # Starts the coroutine up to the first yield

# Send numbers to the coroutine
print(co.send(10))  # Output: 10.0
print(co.send(20))  # Output: 15.0
print(co.send(30))  # Output: 20.0

co.close() # Close the coroutine
```

```
10.0
15.0
20.0

=== Code Execution Successful ===
```

Part 7: Programming (15 points):

- Write a program to initialize an array stock that is initialized with the following stock prices for a share: 22.2, 22.7, 23.5, 22.8, 24.3, 25.6. Then write a function that takes an array and size as a parameter and returns the sum and average (mean) of the array. The means is computed from adding all numbers in the array and divided by the size of the array.
- Write a program to initialize an array called selling that is initialized with the following sell prices for different months per year: 80, 50, 35, 65, 127, 77, 92, 85, 123, 90, 55, 124. Then write a function that takes an array and size as a parameter and returns the largest, second largest and smallest elements.
- Write a program that initializes a matrix called quantity that is initialized with the following numbers:

  Row 0: 2, 4, 3, 6, 9
  Row 1: 5, 8, 9, 3, 7
  Row 2: 1, 4, 3, 2, 10

  Write a function that takes the matrix, rows, columns and row number then reverse that row number in the matrix.

  For example, call the function: reverse(quantity, 3, 5, 1), so the function reverse is called with the matrix quantity, number of rows = 3, number of columns = 5 and row number = 1 to reverse only that specific row.

References

Marangon, J. D. (2024, October 29). *How to Use Coroutines in Python: A Beginner Guide*. Medium, https://medium.com/@johnidouglasmarangon/how-to-use-coroutines-in-python-a-begginer-guide-ba9cfbba12aa.