Abigail Tamburello

Languages & Paradigms

Ch. 9 Assignment

Part 1: Subprograms (15 points)

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

A subprogram is a repeatable sequence of instructions that is called by other locations in a program. Depending on the specific programming language, a subprogram can be called a subroutine, function, method, etc..

Subprograms all have a single entry point. They can only be executed one at a time; they cannot be executed simultaneously. After that, the execution terminates, and the control returns to the caller. Most subprograms are named.

Subprograms are collections of statements that define parameterized computations.

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

Procedures and functions are the two types of subprograms.

Functions return values. Procedures do not return values.

In some cases, functions can be defined to be used as procedures.

Procedures define new statements. Fortran and Ada are older languages that support procedures.

Procedures return results by modifying non-parameter variables or by altering formal parameters. In both cases, the caller observes changes after the procedure is executed.

Functions ideally cause no side effects and return a single value. Functions are sometimes used to modify parameters or external variables. Functions are invoked within expressions, and the returned value replaces the function call itself. Functions act as user-defined operators.

Procedure-like example in Python:

```
counter = 0
def increment():
    global counter
    counter += 1

increment()
print(counter) # Output: 1
```

Function example in Python:

```
def add(a, b):
    return a + b

result = add(3, 4)
print(result) # Output: 7
```

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

Subprograms are able to break a large program into smaller, self-contained units, making the code easier to manage. Subprograms can also be reused in many places. Abstraction is a good part of subprograms because it allows programmers to use functionality without needing to understand the internal systems.

Subprograms can impact the performance because calling subprograms adds call/return overhead. Too many subprograms can fragment the code and make control flow harder to follow.

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.

When a parameter is passed by value, the actual parameter value is used to initialize the corresponding formal parameter, which then acts as a local variable in the subprogram, which in turn implements in-mode semantics. This is normally implemented by copy because this is the fastest approach. An advantage of this is that for scalars it is fast in linkage cost and access time. A disadvantage of this method is that storage could be overloaded if the parameter is too big.

Pass-by-reference is an implementation model for out-mode parameters. Using this method results in no value being transmitted to the program. This has similar advantages/disadvantages to pass-by-value. There is a possibility of an actual parameter collision.

Pass by name is an inout-mode parameter transmission method that does not correspond to a single implementation model. The actual parameter is textually substituted for the corresponding formal parameter in all its occurrences in the subprogram. This one is very different from the other two mentioned. It is not used as part of a widely used language.

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

C showing pass-by-value:

```
#include <stdio.h>

void f(int x) {
    x = x +10;
}

int main() {
    int a = 5;
    f(a);
```

```
        printf(endl, a) // output 5
}
```

C++ pass by reference

```
void g(int &x) {
        x = x + 10;
}
int main() {
        int a = 5;
        g)a);
        cout << a << endl; // output 15
}
```

Scala pass by name

```
def h(x: => Int) = {
  println("Eval 1: " + x)
  println("Eval 2: " + x)
}

var a = 10
h(a + 1)
a = 20
h(a + 1)
```

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

A programmer chooses the best parameter passing mechanism by analyzing the costs and benefits of performance, reliability, conciseness, control, and their specific goals. Value parameters are easy and manageable. Performance is important so that the program can run smoothly without damaging the computer by being overloaded with data. We want our programs to run smoothly and efficiently without damaging anything.

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

Define and explain the concept of function overloading.

Function overloading of subprograms occurs when one operator has multiple meanings. This usually happens when a subprogram has the same name as another subprogram in the same referencing environment. The program would encounter compile errors.

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

C++

```
#include <iostream>
using namespace std;
void print(int x) {
   cout << "Int: " << x << endl;
}
void print(double x) {
   cout << "Double: " << x << endl;
}
int main() {
   print(5);
   print(3.14);
   return 0;
```

```
}
```

Another example in java:

```
public class Math {
  static int add(int a, int b) {
    return a + b;
  }
  static int add(int a, int b, int c) {
    return a + b + c;
  }

  public static void main(String[] args) {
    System.out.println(add(2, 3));
    System.out.println(add(1, 2, 3));
  }
}
```

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

Benefits of using overloaded functions include allowing the same function name to handle different types or numbers of inputs. Polymorphism is a benefit as well. Some challenges include ambiguity or confusion when functions are too similar, and this could lead to more errors and make it harder to maintain the code.

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.

A generic function allows parameters to operate on values of different data types without rewriting the function for each type. The functions uses a placeholder that is filled in when the function is called.

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

In C++ programs, generic functions are more flexible than regular functions. Also, in generic functions, repetition is eliminated because of the generalized logic, while regular functions may require separate functions or overloading for different types.

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

C++

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

template <typename T>
T maxValue(T a, T b) {
   return (a > b) ? a : b;
}
```

```
int main() {
   cout << maxValue(3, 7) << endl;
   cout << maxValue(2.5, 1.8) << endl;
   cout << maxValue('a', 'z') << endl;
}
```

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

template <class T>
void swapValues(T &x, T &y) {
   T temp = x;
   x = y;
   y = temp;
}

int main() {
   int a = 10, b = 20;
   swapValues(a, b);
   string s1 = "hello", s2 = "world";
   swapValues(s1, s2);
```

```
    cout << a << " " << b << endl;
    cout << s1 << " " << s2 << endl;
}
```

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

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

Functions are parameters refers to the ability to pass a function as an argument to another function. This is significant because it allows programmers to create flexible and reusable code. This allows for the creation of paradigms like callbacks, event handling, and functional programming techniques.

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

Python

```
def greet(name):
    return f"Hello, {name}!"

def run_callback(func, value):
    return func(value)

print(run_callback(greet, "Alice"))   # passes greet() as a parameter
```

```
def by_length(word):
    return len(word)

words = ["banana", "passionfruit", "orange"]
sorted_words = sorted(words, key=by_length)

print(sorted_words)   # sorts by string length
```

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

Passing functions as parameters makes programs more flexible because it separates what a piece of code does from how the code is executed. This allows for customization of behavior without modifying the original function. This overall makes code easier to maintain, extend, and adapt to new requirements.

Part 6: Coroutine: (10 points) (read section 9.13)

Discuss the concept of coroutines and its significance in programming.

A coroutine is a type of subprogram that can have multiple entry points, all controlled by the coroutine itself. Caller and called coroutines are more equitable. They have the ability to maintain their status between activations. Usually, coroutines are created in an application by a master unit, so that once created, they can execute their initialization code and then return control to the master unit.

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