**Assignment 5**

Computer Science 350

Programming Language Concepts

Department of Computer Science

University of Regina

Winter 2021

1. (20 Marks)

Briefly describe the following concepts:

Primitive Data Types : The variable’s data type which holds simple values are called Primitive Data Type. Also the major difference it has over other data types is that it contains the value over an address. So there is a duplicate value rather than having the address as the original value.

Character String Types : An array of characters is known as string. String data type is a class which has multiple functions which help manipulate the instance of a string.

User-Defined Ordinal Types : A set of possible values associated using positive integers is known as Ordinal Types. There are two types of Ordinal Types, one is enumeration and the other one is subrange.

Array Types : The element which is formed by multiple instances of its sibling and the main variable holds the address of the first element is known as Array.

Associative Arrays : it is the same as an array but the data elements are unordered and separated using a key.

Record Types : Record types instances whose data elements are identified according to their names. Struct and Union are its examples.

Tuple Types : A tuple can be defined without using any parameters. Also it can have as many items as defined which might be of different types.

List Types : List type is the most versatile data type which is used to store data elements of the same type.

Union Types : An instance which has its elements change and store values in the run time. So if a union instance is created, in the middle of execution there can be some more variables added to this instance.

Pointer and Reference Types : a type in which a value or range of values can be stored in the memory and its memory address is stored in a variable is known as pointer type. Reference type is similar to pointer type but it carries the address of some value from their pointer and manipulates the using the address value. Reference types are mainly used in sending receiving data from a function.

2. (10 marks)

Briefly explain the following concepts:

- Formal parameters and actual parameters

- Positional and keyword methods for Actual/Formal Parameter Correspondence

- Parameter passing methods

Ans:

**Formal Parameter** — the identifier used in a method to stand for the value that is passed into the method by a caller.This are parameters that appear in function calling.

**Actual Parameter** — the actual value that is passed into the method by a caller, or are parameters that appear in function declarations.

They are called "parameters' ' because they define information that is passed to a function.But both are different from one another.For instance,in Actual parameters,there is no compelling reason to determine the datatype.They can be constant values or variable names. However, on the other hand, the extent of formal contentions is local to the function definition. They can be treated as nearby variables of a function in which they are used.

Parameters passing methods are the courses in real boundaries that are transmitted to the called subprogram. The parameters can be passed to a subprogram in a startling manner. The most common ways are pass-by-value and pass-by-reference. Different strategies are used by the programming activity. The parameter passing method are arranged with the objective that the arranged statergy is viable enough in both execution time and memory use. Pass by reference is adequately beneficial if clusters and designs are to be passed.

3. (5 marks) Consider the following program written in C syntax:

void swap(int a, int b) {

int temp;

temp = a;

a = b;

b = temp;

}

void main() {

int value = 2, list[5] = {1, 3, 5, 7, 9};

swap(value, list[0]);

swap(list[0], list[1]);

swap(value, list[value]);

}

For each of the following parameter-passing methods, what are all of the values of the variables value and list after each of the three calls to swap?

Passed by value

No arguments are changed in Pass by values as the value is copied and is a duplicate of the original variable, so after the function returns the variables get back the values they were initialized with.

Passed by reference

The arguments are changed when variables are passed by reference.

After the **first call to swap**,

*value* = 1

*list[0]* = 2

After the **second call to swap**,

*list[0]* = 3

*list[1]* = 2

After the **third call to swap**,

*value* = 2

*list[1]* = 1

Passed by value-result

For this case, If the variables are passed by value-result it will have the same effect as passed by reference.

4. (5 marks) Consider the following program written in C syntax:

void fun (int first, int second) {

first += first;

second += second;

}

void main() {

int list[2] = {1, 3};

fun(list[0], list[1]);

}

For each of the following parameter-passing methods, what are the values of the list array after execution?

Passed by value

No arguments are changed in Pass by values as the value is copied and is a duplicate of the original variable, so after the function returns the variables get back the values they were initialized with.

Passed by reference

The arguments are changed when variables are passed by reference.

After the **function call to fun**,

*list[0]* = 3

*list[1]* = 2

Passed by value-result

For this case, If the variables are passed by value-result it will have the same effect as passed by reference.

5. (10 Marks) Describe an implementation of PASCAL style

subprograms (i.e., nested subprogram) with both dynamic

and static bindings.

Ans: When a routine is added inside of another function or procedure this type of program is a nested styled program. In the Nested procedure subprogram a frame pointer address of the parameter is passed to the nested routine. Doing this local variables and parameters of the calling routine can be accessed using nested routine

6. (10 Marks) Show the stack with all activation record instances, including static and dynamic chains, when execution reaches position 1 in the following skeletal program. Assume bigsub is at level 1.

function bigsub() {

function a() {

function b() {

... <----------------------------1

} // end of b

function c() {

...

b();

...

} // end of c

...

c();

...

} // end of a

...

a();

...

} // end of bigsub

Ans:

**Static Representation**

Function b() -> Function bigsub() -> Function a() -> Function b() -> Function c() -> Function b() -> Function a()

**Other Representation**

