Unit I

CHAPTER-1 FUNCTION

Book Back Evaluations

Part - I
Choose the best answer (1 Mark)

1. The small sections of code that are used to perform a particular task is called

(A) Subroutines (B) Files (C) Pseudo code (D) Modules

2. Which of the following is a unit of code that is often defined within a greater code structure?

(A) Subroutines **(B) Function** (C) Files (D) Modules

3. Which of the following is a distinct syntactic block?

(A) Subroutines (B) Function **(C) Definition** (D) Modules

4. The variables in a function definition are called as

(A) Subroutines (B) Function (C) Definition **(D) Parameters**

5. The values which are passed to a function definition are called

(A) Arguments (B) Subroutines (C) Function (D) Definition

6. Which of the following are mandatory to write the type annotations in the function definition?

(A) Curly braces **(B) Parentheses** (C) Square brackets (D) indentations

7. Which of the following defines what an object can do?

(A) Operating System (B) Compiler **(C) Interface** (D) Interpreter

8. Which of the following carries out the instructions defined in the interface?

(A) Operating System (B) Compiler **(C) Implementation** (D) Interpreter

9. The functions which will give exact result when same arguments are passed are called

(A) Impure functions (B) Partial Functions (C) Dynamic Functions **(D) Pure functions**

10. The functions which cause side effects to the arguments passed are called

(A) impure function (B) Partial Functions (C) Dynamic Functions (D) Pure functions

Part - II
Answer the following questions (2 Marks)

1. **What is a subroutine?**

Subroutines are small sections of code that are used to perform a particular task that can be used repeatedly

2. **Define Function with respect to Programming language**.

- A function is a unit of code that is oft en defined within a greater code structure.
- Specifically, a function contains a set of code that works on many kinds of inputs, like variants, expressions and produces a concrete output.

3. **Write the inference you get from X:=(78).**

X:= (78) has an expression in it but (78) is not itself an expression.

(78) is a function definition.

Definitions bind values to names, in this case the **value 78** being bound to the name *'X'*.

4. Differentiate interface and implementation.

Interface	Implementation
Interface just defines what an object can do, but won't actually do it	Implementation carries out the instructions defined in the interface

5. Which of the following is a normal function definition and which is recursive function definition

i) let rec sum x y:
return x + y → **recursive function definition**

ii)let disp :
print 'welcome' → **normal function definition**

iii) let rec sum num:
if (num!=0) then rez.turn num + sum (num-1)
else
return num → **recursive function definition**

Part - III

Answer the following questions (3 Marks)

1. Mention the characteristics of Interface.

- The class template specifies the interfaces to enable an object to be created and operated properly.
- An object's attributes and behavior is controlled by sending functions to the object.

2. Why strlen is called pure function?

" Pure functions are functions which will give exact result when the same arguments are passed "

- Strlen(s) is called each time and strlen needs to iterate over the whole of 's'. If the compiler is smart enough to work out that strlen is a pure function and that ***'s'*** is not updated in the loop.

3. What is the side effect of impure function? Give example.

The variables used inside the function may cause side effects though the functions which are not passed with any arguments. In such cases the function is called impure function

Example :

```
let y: = 0
(int) inc (int) x
y: = y + x;
return (y)
```

4. Differentiate pure and impure function.

Pure Function	Impure Function
The return value of the pure functions solely depends on its arguments passed. Hence, if you call the pure functions with the same set of arguments, you will always get the same return values. They do not have any side effects.	The return value of the impure functions does not solely depend on its arguments passed. Hence, if you call the impure functions with the same set of arguments, you might get the different return values For example, random(), Date().
They do not modify the arguments which are passed to them	They may modify the arguments which are passed to them

5. What happens if you modify a variable outside the function? Give an example.

- One of the most popular groups of side effects is modifying the variable outside of function.

For example

```
let y: = 0
(int) inc (int) x
y: = y + x;
return (y)
```

In the above example the value of y get changed inside the function definition due to which the result will change each time.

The side effect of the inc () function is it is changing the data of the external visible variable **'y'**. As you can see some side effects are quite easy to spot and some of them may tricky.

A good sign that our function impure *(has side effect)* is that it doesn't take any arguments and it doesn't return any value.

Part - IV
Answer the following questions (5Marks)

1. What are called Parameters and write a note on

i) Parameter without Type (ii) Parameter with Type

Parameters (and arguments)

- **Parameters** are the variables in a function definition.
- **Arguments** are the values which are passed to a function definition.

(i) Parameter without Type

```
(requires: b>=0 )
(returns: a to the power of b)
let rec pow a b:=
if b=0 then 1
else a * pow a (b-1)
```

⟹ *a, b are declared without Data type.*

- In the above function definition variable *'b'* is the parameter and the value which is passed to the variable *'b'* is the argument.
- The precondition ***(requires)*** and postcondition ***(returns)*** of the function is given. Note we have not mentioned any types: ***(data types)***.
- Some language compiler solves this type ***(data type)*** inference problem algorithmically, but some require the type to be mentioned.

(ii) Parameter with Type

```
requires: b> 0 )
(returns: a to the power of b )
let rec pow (a: int) (b: int) : int :=
if b=0 then 1
else a * pow b (a-1)
```

⟹ *a, b are declared with Data type (int)*

- When we write the type annotations for ***'a'*** and ***'b'*** the parentheses are mandatory. Generally we can leave out these annotations, because it's simpler to let the compiler infer them.
- There are times we may want to explicitly write down types. This is useful on times when you get a type error from the compiler that doesn't make sense.
- Explicitly annotating the types can help with debugging such an error message.

2. Identify in the following program

```
let rec gcd a b :=
  if b <> 0 then gcd b (a mod b) else return a
```

i) Name of the function **gcd**

ii) Identify the statement which tells it is a recursive function

Yes , the variable *'b'* becomes *'0'* is recursive function.

iii) Name of the argument variable **a and b**

iv) Statement which invoke the function recursively **(a mod b) until *'b'* becomes *'0'***

v) Statement which terminates the recursion **When variable *'b'* becomes *'0'* terminated**

3. Explain with example Pure and impure functions.

Pure functions:

Pure functions *are functions which will give exact result when the same arguments are passed.*

For example the mathematical function sin (0) always results 0. This means that every time you call the function with the same arguments, you will always get the same result. A function can be a pure function provided it should not have any external variable which will alter the behaviour of that variable.

Let us see an example

```
let square x
return: x * x
```

The above function square is a pure function because it will not give different results for same input.

Impure functions:

The variables used inside the function may cause side effects though the functions which are not passed with any arguments. In such cases the function is called impure function.

When a function depends on variables or functions outside of its definition block, you can never be sure that the function will behave the same every time it's called.

For example the mathematical function random() will give different outputs for the same function call.

```
let Random number
let a := random()
if a > 10 then
return: a
else
return: 10
```

Here the function **Random is impure** as it is not sure what will be the result when we call the function.

4. Explain with an example interface and implementation.

Interface is a set of action that an object can do. For example when you press a light switch, the light goes on, you may not have cared how it splashed the light. In Object Oriented Programming language, an Interface is a description of all functions that a class must have in order to be a new interface.

In our example, anything that ***"ACTS LIKE"*** a light, should have function definitions like turn_on () and a turn_off (). The purpose of interfaces is to allow the computer to enforce the properties of the class of ***TYPE T*** *(whatever the interface is)* must have functions called *X, Y, Z,* etc.

Implementation: A class declaration combines the external interface *(its local state)* with an implementation of that interface *(the code that carries out the behaviour)*. An object is an instance created from the class.

The interface defines an object's visibility to the outside world.

Let us see a simple example, consider the following implementation of a function that finds the minimum of its three arguments:

```
let min 3 x y z :=
if x < y then
if x < z then x else z
else
if y < z then y else z
```

KEY POINTS TO REMEMBER:

- Algorithms are expressed using statements of a programming language.

• Subroutines are small sections of code that are used to perform a particular task that can be used repeatedly.

• A function is a unit of code that is often defined within a greater code structure.

• A function contains a set of code that works on many kinds of inputs and produces a concrete output.

• Definitions are distinct syntactic blocks.

• Parameters are the variables in a function definition and arguments are the values which are passed to a function definition through the function definition.

• When you write the type annotations the parentheses are mandatory in the function definition.

• An interface is a set of action that an object can do.

• Interface just defines what an object can do, but won't actually do it.

• Implementation carries out the instructions defined in the interface.

• Pure functions are functions which will give exact result when the same arguments are passed.

• The variables used inside the function may cause side effects though the functions which are not passed with any arguments. In such cases the function is called impure function.

SASTRA MATRICULATION HIGHER SECONDARY SCHOOL,

KILPENNATHUR, TIRUVANNAMALAI 604 601

Mobile No : 9655826843 ***Email : vijay28soft@gmail.com***

M.VIJAYA KUMAR, MCA.,M.Phil.,B.Ed.,PGDCA.,

PGT-COMPUTER TEACHER