UNIT-I PROBLEM SOLVING TECHNIQUES

CHAPTER 1 FUNCTION

Public Exam Frequently Asked Questions

1 MARK

CHOOSE THE BEST ANSWER

1. The small sections of code that are used to perform a particular task is called *[Aug-2021; FRT-'22]*

(a) Subroutines (b) Files
(c) Pseudo code (d) Modules

[Ans. (a) Subroutines]

2. Which of the following is a unit of code that is often defined within a greater code structure? *[July-'22]*

(a) Subroutines (b) Function
(c) Files (d) Modules

[Ans. (b) Function]

3. Which of the following is a distinct syntactic block? *[PTA-6; FRT & May-'22]*

(a) Subroutines (b) Function
(c) Definition (d) Modules

[Ans. (c) Definition]

4. The variables in a function definition are called as *[PTA-2; QY-2019]*

(a) Subroutines (b) Function
(c) Definition (d) Parameters

[Ans. (d) Parameters]

5. The values which are passed to a function definition are called *[HY-2019; FRT-'22]*

(a) Arguments (b) Subroutines
(c) Function (d) Definition

[Ans. (a) Arguments]

6. Which of the following are mandatory to write the type annotations in the function definition? *[PTA-4; FRT-'22]*

(a) Curly braces (b) Parentheses
(c) Square brackets (d) Indentations

[Ans. (b) Parentheses]

7. The functions which will give exact result when same arguments are passed are called *[PTA-3; Mar.-2020]*

(a) Impure functions (b) Partial Functions
(c) Dynamic Functions (d) Pure functions

[Ans. (d) Pure functions]

8. A function definition which call itself : *[PTA-1]*

(a) Pure function (b) Impure function
(c) Normal function
(d) Recursive function

[Ans. (d) Recursive function]

9. Which is the basic building block of computer programs? *[Sep-2020]*

(a) Argument (b) Parameter
(c) Subroutine (d) Interface

[Ans. (c) Subroutine]

2 MARKS

ANSWER THE FOLLOWING QUESTIONS

1. What is a subroutine? *[PTA-1; HY-2019]*

Ans. **(i)** Subroutines are the basic building blocks of computer programs. Subroutines are small sections of code that are used to perform a particular task that can be used repeatedly.

(ii) In Programming languages these subroutines are called as Functions.

2. Define Function with respect to Programming language. *[Aug-2021; FRT-'22]*

Ans. A function is a unit of code that is often 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. Define pure function. Give one example. *[QY-2019]*

Ans.
```
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
```

3 MARKS

Answer the following questions

1. Mention the characteristics of Interface. *[Sep-2020]*

Ans. **(i)** The class template specifies the interfaces to enable an object to be created and operated properly.

(ii) An object's attributes and behaviour is controlled by sending functions to the object.

2. Why strlen is called pure function? *[Govt. MQP-2019]*

Ans. **(i)** strlen is a pure function because the function takes one variable as a parameter, and accesses it to find its length.

(ii) This function reads external memory but does not change it, and the value returned derives from the external memory accessed.

3. What is the side effect of impure function. Give example. *[PTA-5]*

Ans. Impure Function has the following side effects

(i) Function impure (has side effect) is that it doesn't take any arguments and it doesn't return any value.

(ii) Function depends on variables or functions outside of its definition block.

(iii) It never assure you that the function will behave the same every time it's called.

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

(iv) Here, the result of inc() will change every time if the value of 'y' get changed inside the function definition.

(v) Hence, the side effect of inc () function is changing the data of the external variable 'y'.

4. Differentiate pure and impure function.

Ans. *[PTA-3, 6; Mar.-2020]*

S. No.	Pure	Impure
(i)	The return value of the pure functions solely depends on its arguments passed.	The return value of the impure functions does not solely depend on its arguments passed.
(ii)	If you call the pure functions with the same set of arguments, you will always get the same return values.	If you call the impure functions with the same set of arguments, you might get the different return values.
(iii)	They do not have any side effects. For example: strlen(), sqrt()	They have side effects. For example: random(), Date().
(iv)	They do not modify the arguments which are passed to them	They may modify the arguments which are passed to them

5. Write a function that finds the minimum of its three arguments. *[PTA-4; QY-2019]*

Ans.
```
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
```

5 MARKS

Answer the following questions

1. What are called Parameters and write a note on *[PTA-2; May-'22]*

(i) Parameter without Type *[FRT-'22]*

(ii) Parameter with Type

Ans. **Parameters (and arguments) :** Parameters are the variables in a function definition and arguments are the values which are passed to a function definition.

(i) Parameter without Type : Let us see an example of a function, definition :

```
(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)
```

- 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.
- In the above function definition if expression can return 1 in the then branch, shows that as per the **typing** rule the entire if expression has type **int**.
- Since the if expression is of type **'int'**, the function's return type also be 'int'. **'b'** is compared to 0 with the equality operator, so **'b'** is also a type of 'int'. Since 'a' is multiplied with another expression using the * operator, **'a'** must be an int.

(ii) Parameter with Type : Now let us write the same function definition with types for some reason:

```
(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)
```

- 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 *[PTA-5]*

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

i) Name of the function
ii) Identify the statement which tells it is a recursive function
iii) Name of the argument variable
iv) Statement which invoke the function recursively
v) Statement which terminates the recursion

Ans. **(i)** gcd
(ii) let rec gcd
(iii) a, b
(iv) gcd b (a mod b)
(v) return a

3. Explain with example Pure and impure functions.

Ans. **Pure functions :**

(i) Pure functions are functions which will give exact result when the same arguments are passed.

(ii) 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.

(iii) 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
```

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

(v) There are various theoretical advantages of having pure functions. One advantage is that if a function is pure, then if it is called several times with the same arguments, the compiler only needs to actually call the function once. Lt's see an example

```
let i: = 0;
  if i <strlen (s) then
  -- Do something which doesn't affect s
  ++i
```

(vi) If it is compiled, 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, then it can remove the redundant extra calls to strlen and make the loop to execute only one time.

(vii) From these what we can understand, strlen is a pure function because the function takes one variable as a parameter, and accesses it to find its length. This function reads external memory but does not change it, and the value returned derives from the external memory accessed.

Impure functions :

(i) 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.

(ii) 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 randomnumber:=
    a := random()
      if a > 10 then
        return: a
else
      return: 10
```

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

☆☆☆