DO IT NOW!!! SOMETIMES LATER BECOMES NEVER!!!

START YOUR PREPARATION NOW AND SUCCEED IN YOUR EXAMS!

Namma Kalvi
www.nammakalvi.in

MATERIALS FOR

XII- COMPUTER SCIENCE

J. BASKARAN M.Sc., B.Ed. (C.S)
jbaskaran89@gmail.com
Puducherry.

J. ILAKKIA M.Sc., M.Phil., B.Ed. (C.S)
jilakkia@gmail.com
Puducherry.

J. BASKARAN M.Sc., B.Ed. (C.S)
jbaskaran89@gmail.com
Puducherry.

J. ILAKKIA M.Sc., M.Phil., B.Ed. (C.S)
jilakkia@gmail.com
Puducherry.

COMPUTER SCIENCE

1. FUNCTIONS

Section – A

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 functions (B) Partial Functions

(C) Dynamic Functions (D) Pure functions

Section-B

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 often defined within a greater code structure.
- A function works on many kinds of inputs and produces a concrete output

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

- **X:=(78)** is a function definition.
- Definitions bind values to names.
- Hence, the value **78** 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

Ans: Recursive Function

ii) let disp :

print 'welcome'

Ans: Normal Function

iii) let rec sum num:

if (num!=0) then return num + sum (num-1)

else

return num

Ans: Recursive Function

Section-C

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 behaviour is controlled by sending functions to the object.

2. Why strlen is called pure function?

- **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.

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

- Impure Function has the following side effects,
  - Function impure *(has side effect)* is that it doesn't take any arguments and it doesn't return any value.
  - Function depends on variables or functions outside of its definition block.
  - It never assure you that the function will behave the same every time it's called.
- **Example:**

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

  - Here, the result of ***inc()*** will change every time if the value of ***'y'*** get changed inside the function definition.
  - Hence, the side effect of inc () function is changing the data of the external variable ***'y'.***

4. Differentiate pure and impure function.

Pure Function	Impure Function
• Pure functions will give exact result when the same arguments are passed.	• Impure functions never assure you that the function will behave the same every time it's called.
• Pure function does not cause any side	• Impure function causes side effects to

effects to its output.	its output.
• 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.
• They do not modify the arguments which are passed to them.	• They may modify the arguments which are passed.
• Example: strlen(), sqrt()	• Example: random(), Date()

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

- Modifying the variable outside of function causes side effect.
- Example:

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

  - Here, the result of ***inc()*** will change every time if the value of ***'y'*** get changed inside the function definition.
  - Hence, the side effect of inc () function is changing the data of the external variable ***'y'.***

Section - D

Answer the following questions: **(5 Marks)**

1. What are called Parameters and write a note on

(i) Parameter without Type (ii) Parameter with Type

Answer:

- **Parameters** are the variables in a function definition
- **Arguments** are the values which are passed to a function definition.
- Two types of parameter passing are,
  1. Parameter Without Type
  2. Parameter With Type

1. Parameter Without Type:

- Lets see an example of a function definition of 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)
```

- In the above function definition **variable ' *b'*** is the **parameter** and the **value** passed **to** the variable ***'b'*** is the **argument**.

- The precondition ***(requires)*** and postcondition ***(returns)*** of the function is given.

- We have not mentioned any types: ***(data types)***. This is called parameter without type.

- In the above function definition the expression has type ***'int',*** so the function's return type also be ***'int'*** *by implicit*.

2. Parameter With Type:

- Now let us write the same function definition with types,

```
(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)
```

- In this example we have explicitly annotating the types of argument and return type as ***'int'***.
- Here, when we write the type annotations for 'a' and 'b' the parantheses are mandatory.
- This is the way passing parameter with type which helps the compiler to easily infer them.

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

- ***let rec gcd a b :=***
- **"rec" keyword tells the compiler it is a recursive function**
-

iii) Name of the argument variable

- **'a'** and **'b'**

iv) Statement which invoke the function recursively

- ***gcd b (a mod b)***

v) Statement which terminates the recursion

- ***return a***

3. Explain with example Pure and impure functions.

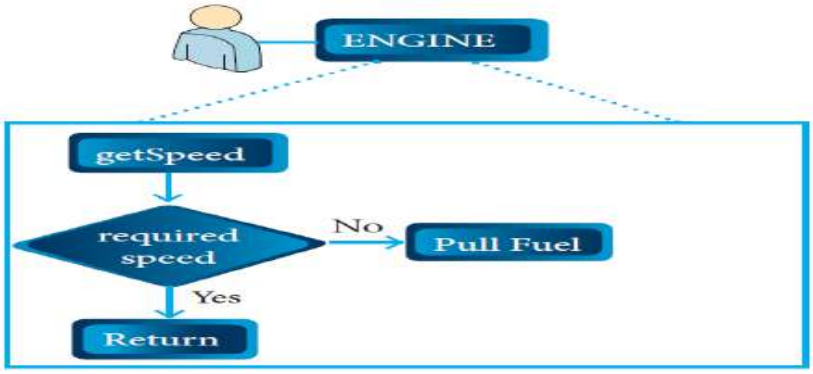
Pure Function	Impure Function
• Pure functions will give exact result when the same arguments are passed.	• Impure functions never assure you that the function will behave the same every time it's called.
• Pure function does not cause any side effects to its output.	• Impure function causes side effects to its output.
• 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.
• They do not modify the arguments which are passed to them	• They may modify the arguments which are passed.
• If we call pure functions with same set of arguments, we will always get the same return values.	• If we call impure functions with same set of arguments, we might get the different return values.

Example: sqrt()
let square x
return: x * x
	• **Example: random()**
	let Random number
	let a := random()
	if a > 10 then
	return: a
	else
	return: 10

4. Explain with an example interface and implementation.

- **Interface**
  - 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.
  - The interface defines an object's visibility to the outside world.
  - In Object Oriented Programming language, an Interface is a description of all functions that a class must have.
  - The purpose of interfaces is to allow the computer to enforce the properties of the class which means the class of ***TYPE T*** *(whatever the interface is)* must have functions called *X, Y, Z,* etc.
  - For example when you press a light switch, the light goes on, you may not have cared how it splashed the light
  - In our example, anything that ***"ACTS LIKE"*** a light, should have function definitions like turn_on () and a turn_off ().
  - An object ***"ACTS LIKE"*** is an instance created from the class ***"LIGHT"***. All the objects of class ***"LIGHT"*** will uses all its functions.
- **Characteristics of interface:**
  - The class template specifies the interfaces to enable an object to be created and operated properly.
    - An object's attributes and behaviour is controlled by sending functions to the object.
- **Implementation:**
  - Implementation carries out the instructions defined in the interface
  - How the object is processed and executed is the implementation.
  - A class declaration combines the external interface *(its local state)* with an implementation of that interface *(the code that carries out the behaviour)*.
- **Example:**

Let's take the example of increasing a car's speed.



- The person who drives the car doesn't care about the internal working.
- To increase the speed of the car he just presses the accelerator to get the desired behaviour.

- Here the accelerator is the interface between the driver *(the calling / invoking object)* and the engine *(the called object)*.
- In this case, the function call would be Speed (70): This is the interface.
- Internally, the engine of the car is doing all the things.
- It's where fuel, air, pressure, and electricity come together to create the power to move the vehicle.
- All of these actions are separated from the driver, who just wants to go faster.
- Thus we separate interface from implementation.

PREPARED BY

J. BASKARAN M.Sc., B.Ed. (C.S)
jbaskaran89@gmail.com

J. ILAKKIA M.Sc., M.Phil., B.Ed. (C.S)
jilakkia@gmail.com