QB365 Question Bank Software Study Materials

Function Important 2,3 & 5 Marks Questions With Answers (Book Back and Creative)

12th Standard

Computer Science

Total Marks : 75

2 Marks

10 x 2 = 20

1) What is a subroutine?

Answer : (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.

Answer : 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 variables, expressions and produces a concrete output.

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

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

Answer :

INTERFACE	**IMPLEMENTATION**
Interface just defines what an object can do, but won't actually do it.	Implementation carriers 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

ii) let disp:
    print 'welcome'

iii) let rec sum num:
    if (num!=0) then return num + sum (num-1)
    else
    return num

Answer : (i) Recursive function
(ii) Normal function
(iii) Recursive function

6) What are parameters and aruguments?

Answer : Parameters are the variables in a function definition. Arguments are the values which are passed to a function definition.

7) Give an example of function definition parameter without type.

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

What is recursive function?

8) **Answer :** A function definition which call itself is called recursive function.

9) Construct on algorithm that arranges meetings between these two types so that they change their color to the third type. In the end, all should display the same color.

Answer : let rec monochromatize a b c :=

if a > 0 then

a, b, c:= a-1, b-1, c+2

else

a:=0, b:=0, c:= a + b + c

return c

10) What are the two types of function specification?

Answer : The two types of function specification are

(i) Parameter without type.

(ii) parameter with type.

3 Marks 10 x 3 = 30

11) Mention the characteristics of Interface.

Answer : (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.

12) Why strlen is called pure function?

Answer : (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.

13) What is the side effect of impure function. Give example.

Answer : 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.

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
```

14) Differentiate pure and impure function.

Answer :

S.NO	PURE FUNCTION	IMPURE FUNCTION
(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.	They cause 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

15) Wha happens if you modify a variable outside the function? Give an example.

Answer : 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.

16) Write algorithm to find minimum of its three arguments.

Answer : Let min 3 x y z:= If x If x else if y < z then y else z

17) Explain the syntax of function definitions.

Answer : (i) The syntax to define functions is close to the mathematical usage: the definition is introduced by the keyword let, followed by the name of the function and its arguments; then the formula that computes the image of the argument is written after an = sign. If you want to define a recursive function: use **"let rec" instead of "let"**.

(ii) **Syntax:** The syntax for function definitions:

let rec fn a1 a2 ... an :;= k

(iii) Here the 'fn' is a variable indicating an identifier being used as a function name. The names 'a1' to 'an' are variables indicating the identifiers used as parameters. The keyword 'rec' is required if 'fn' is to be a recursive function; otherwise it may be omitted.

18) Write an algorithm to check whether the entered number is even or odd.

Answer : (requires: x > = 0)

let rec even x :=

x=0II odd (x-I)

return 'even'

(requires: x > = 0)

let odd x :=

x< > 0 && even (x-I)

return 'odd'

19) Write a program to check the given number is even or odd.

Answer : **Program:**

(requires:x > = 0)

let rec even x: =

x= 0||odd (x-1)

return 'even'

(requires: x > = 0)

let odd x: =

x< > &&even (x-1)

return 'odd'

20) What are the various syntax for function types.

Answer : The syntax for function types:

$^{*}x \rightarrow y \quad ^{*}xl \rightarrow x2 \rightarrow y \quad ^{*}xl \rightarrow \ldots \rightarrow xn \rightarrow y$

The 'x' and 'y' are variables indicating types. The type x → y is the type of a function that gets an input of type 'x' and returns an output of type 'y'. Whereas x1 → x2 → y is a type of a function that takes two inputs, the first input is of type 'x1' and the second input of type 'x2', and returns an output of type 'y'.

Likewise x1 → ... → xn → y has type 'x' as input of n arguments and 'y' type as output.

5 Marks 5 x 5 = 25

21) What are called Parameters and write a note on

(i) Parameter without Type

(ii) Parameter with Type

Answer : **Parameters and arguments:**

Parameters are the variable 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*powa(b-1)

(i) 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.

(ii) Note we have not mentioned any types **(data types)**. Some language computer solves this type **(data type)** inference problem algorithmically, but some require the type to be mentioned.

(iii) In, the above function deinition if expression can return 1 in the then branch, shows that as per the **typing** rule the entire if expression has type **int**

(iv) 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 reasons:

(requires: b > 0)

(returns: a to the power of b)

let rec pow (a:int) (b:int): int :=

if b=0 then 1

n else a * pow b (a-1)

(i) 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.

(ii) 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.

(iii) The syntax to define functions is close to the mathematical usage: the definition is introduced by the keyword **let** followed by the **name** of the function and its arguments; then the formula that computes the image of the argumentis written atter an := sign. If you want to define a recursive function: use "**let rec**" **instead of** "**let**"

Syntax:The syntax for function detinitions: **let irec fn al a2 ... an := k**

(iv) Here the '**fn**' is used as a function name. The nanmes '**a1**' to '**an**'are variables used as parameters. The keyword '**rec**' is required if '**fn**' is to be a recursive function; otherwise it may be omitted.

22) 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

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

Answer : (i) gcd

(ii) let rec gcd

(iii) a, b

(iv) gcd(a mod b)

(v) return a

23) Explain with example Pure and impure functions.

Answer : 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 wil 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 behavior 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.

Example:

```
let length s:=
    i:= 0
    let i:= 0;
    if i< strlen (s) then
    -- Do something which doesn't affect s
    ++1
```

(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 through the functions which are not passed with any arguments. In which cases the function is called impure function.

(ii) When a function depends on variable 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 functions random ( ) will give different outputs for the same function call.

Example:

```
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

24) Explain with an example interface and implementation.

Answer : **Interface:**

(i) An 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 Programnming language, an Interface is a description of all functions.

(ii) 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.

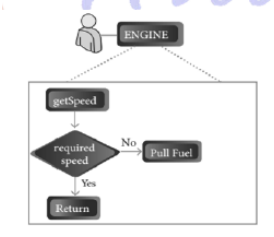
Implementation:

(i) Implementation carries out the instructions defined in the interface.

(ii) How the object is processed and executed is the implementation.

(iii) A class declaration combines the external interface (its local state) with an implementation of that interface (the code that carries out the behaviour).

For example, let's take the example of increasing a car's speed.



(iv) 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).

(v) 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 but fuel, air, pressure, and electricity come together to create the power to move the vehicle.

(vi) All of these actions are separated from the driver, who just wants to go faster. Thus we separate interface from implementation.

25) Explain in detail about chameleons of chromeland problem using function.

Answer : Consider two types of chameleons which are equal in number. construct an algorithm that arranges meetings between these two types so that they change their color to the third type. In the end, all shoultl display the same color.

Let us represent the number of chameleons of each type by variables a, b and c, and their initial values by A, B and C respectively. Let a = b be the input property.

The input output relation is a = b ; 0 and c = A + B + c. Let us name the algorithm Monochromatize The algorithm can be specified as Monochromatize (a,b,c)

--inputs	:	a=A, b=B, c=C, a=b
--outputs	:	a=b=0, c=A+B+C

In each iterative step, two chameleons of the two types (equal in number) meet and change their colors to the third one.

For example, if A, B, C = 4, 4, 6 then the series of meeting will result in

Iteration	a	b	c
0	4	4	4
1	3	3	8
2	2	2	10
3	1	1	12
4	0	0	14

In each meeting, a and b each decreases by 1, and c increased by 2. The solution can be expressed as an iterative algorithm.

Monochromatize (a, b, c)

--inputs:a = A, b = B, c = C, a = b

--outputs: a = b = 0, c = A + B + C

while a > 0

a, b, c:=a-1, b-1, c+2

The algorithm is depicted in the flowchart as below: