HSC SECOND YEAR – COMPUTER SCIENCE

CHAPTER 1: FUNCTION

Prepared by,

J. KAVITHA, B.Sc,B.Ed,M.C.A,M.Phil.,

Computer Instructor Gr - I,

GHSS, S.S.Kulam,

Coimbatore.

https://www.kavikalvi.freeweb.co.in/

Learning Objectives

To Know About,

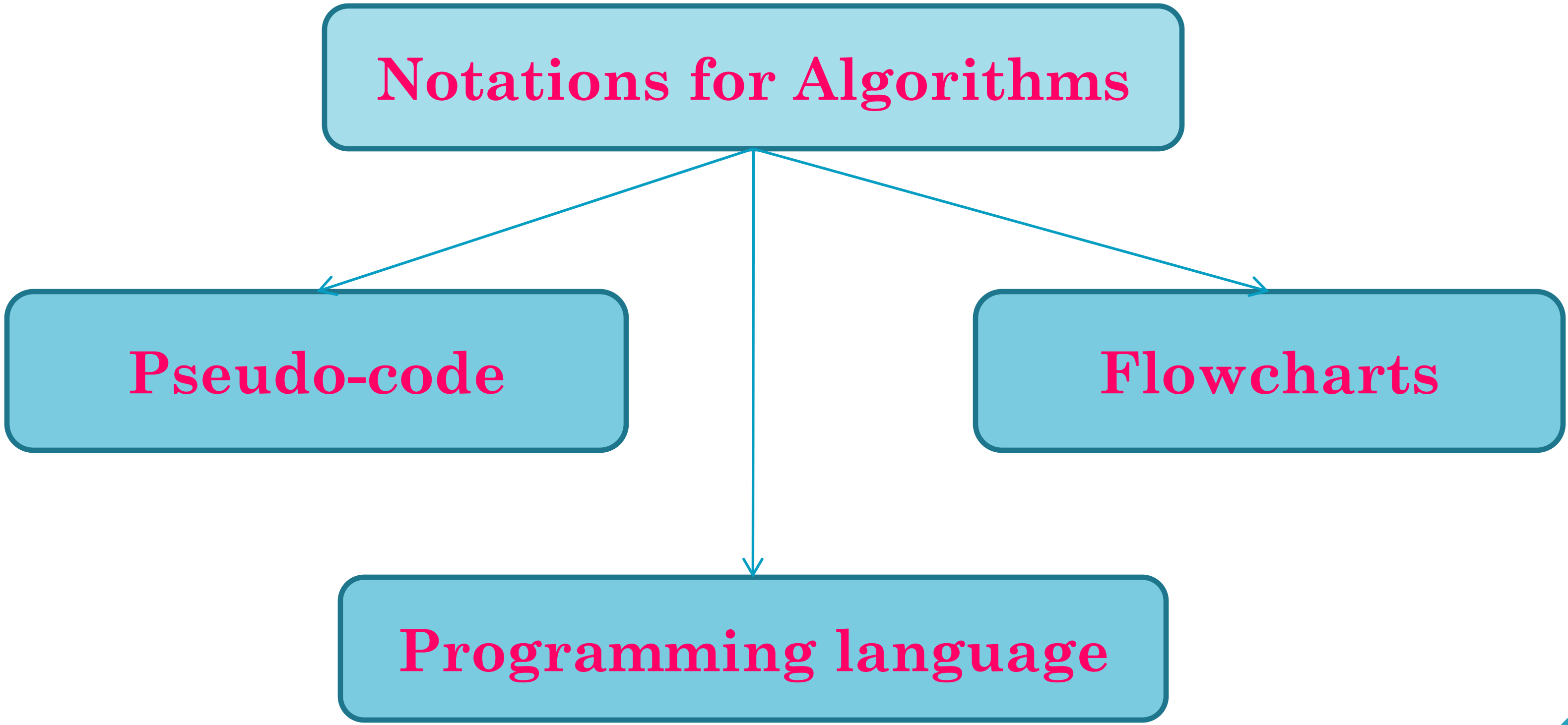
- **Understand Function Specification**
- **Parameters and arguments**
- **Interface Vs Implementation**
- **Pure functions & impure functions**

Algorithms

- Algorithm is a Theoretical representation of a program.
- Algorithm, is a set of step-by-step instructions that define how a work is to be executed to get the expected results.

Notations for Algorithms

- We need a notation to represent algorithms. There are mainly **three different notations** for representing algorithms.

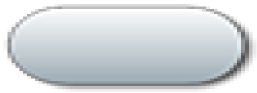
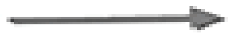
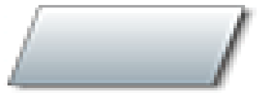
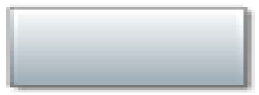
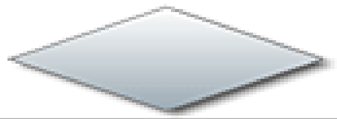


Pseudo code

- Pseudo code is a notation similar to programming languages.
- Algorithms expressed in pseudo code are not intended to be executed by computers, but for communication among people.

Flowchart

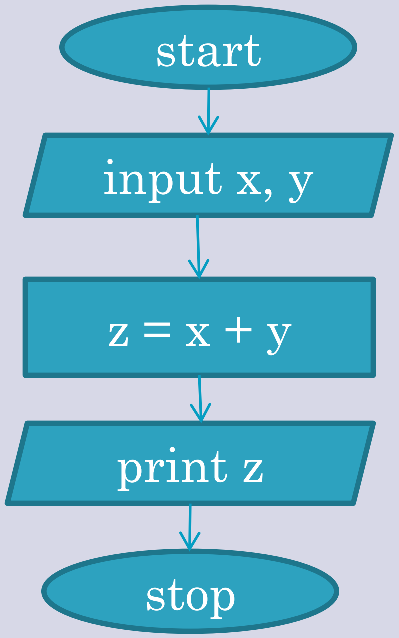
- Flowchart is a diagrammatic notation for representing algorithms. They give a visual intuition of the flow of control, when the algorithm is executed.

Symbol	Name
	Start/end
	Arrows
	Input/Output
	Process
	Decision

Programming Language

- A programming language is a notation for expressing algorithms so that a computer can execute the algorithm.
- An algorithm expressed in a programming language is called a program.
- C, C++ and Python are examples of programming languages.

EXAMPLE: ADDING WO NUMBERS

Algorithms	Pseudo code	Flowchart	Coding
Step1: Start Step2: Get the two numbers Step3: Add those two numbers Step4: Write the answer Step5: Stop	input x , y z = x + y Print z		**x = int(input())** **y = int(input))** **z = x + y** **print(z)**

Functions – Introduction

- The most important criteria in writing and evaluating the algorithm is the time it takes to complete a task.
- Algorithms are expressed using statements of a programming language.
- If a bulk of statements to be repeated for many numbers of times then subroutines are used to finish the task.

SUBROUTINES

- 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.
- In Programming languages these subroutines are called as Functions.

Function with respect to Programming language

- 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.

Function Specification

- Let us consider the example a:= (24).
- a:= (24) has an expression in it but (24) is not itself an expression. Rather, it is a **function definition.**
- Definitions bind values to names, in this case the value 24 being bound to the name '*a*'.
- Definitions are distinct syntactic blocks.

The syntax for function definitions

let rec fn a1 a2 … an := k

- Here the ***'fn' is used as a function name.***
- ***The names '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.***

Example

```
let max x y =
if x>y then
       x
else
       y
```

Parameters and arguments

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

Parameter without Type

- Some language compilers solve this data type problem algorithmically, if we do not specify the data types of variables in the function.

Example

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

- 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 ‘a’ is multiplied with another expression using the * operator, ‘a’ must be an int.

Parameter with Type

Example:

```
(#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.**

RECURSIVE FUNCTION

- A function definition which call itself is called **recursive function.**
- Which of the following is a normal function definition and which is recursive function definition.

```
let sum x y:
    return x + y
```

Normal function

```
let disp :
    print ‘welcome’
```

Normal function

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

Recursive function

Identify in the following program

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

- Name of the function - gcd
- Identify the statement which tells it is a recursive function - let rec gcd a b :=
- Name of the argument variable - a, b
- Statement which invoke the function recursively - gcd b(a mod b)
- Statement which terminates the recursion - return a

Interface Vs Implementation

- An interface is a set of action that an object can do.
- Implementation carries out the instructions defined in the interface
- In object oriented programs classes are the interface and how the object is processed and executed is the implementation.

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
```

The difference between 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

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.

PURE FUNCTIONS

- **Pure functions are functions which will give exact result when the same arguments are passed.**
- **A function can be a pure function provided it should not have any external variable which will alter the behaviour of that variable.**

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.

எடுத்துக்காட்டு

```
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.

Modify variable outside a function

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

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’.

Differentiate pure and impure function

Pure function	Impure function
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.
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.
They do not have any side effects.	They have side effects.
They do not modify the arguments which are passed to them.	They may modify the arguments which are passed to them.

EVALUATION

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

   (A) Subroutines

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

   (B) Function

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

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

   (C) Definition

4. The variables in a function definition are called as

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

   (D) Parameters

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

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

   (A) Arguments

EVALUATION

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

   (A) { }   (B) ( )   (C) [ ]   (D) < >

   (B) ( )

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

   (A) Operating System   (B) Compiler

   (C) Interface   (D) Interpreter

   (C) Interface

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

   (A) Operating System   (B) Compiler

   (C) Implementation   (D) Interpreter

   (C) Implementation

EVALUATION

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

   (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

   (A) Impure functions

Important questions:

1. What is a subroutine?
2. Define Function with respect to Programming language.
3. Differentiate interface and implementation.
4. Mention the characteristics of Interface.
5. Why strlen is called pure function?
6. Differentiate pure and impure function.
7. What are called Parameters?
8. Write a note on: (i) Parameter without Type
   (ii) Parameter with Type
9. Explain with example Pure and impure functions.
10. Explain with an example interface and implementation.

THANK YOU!!!

Education is the

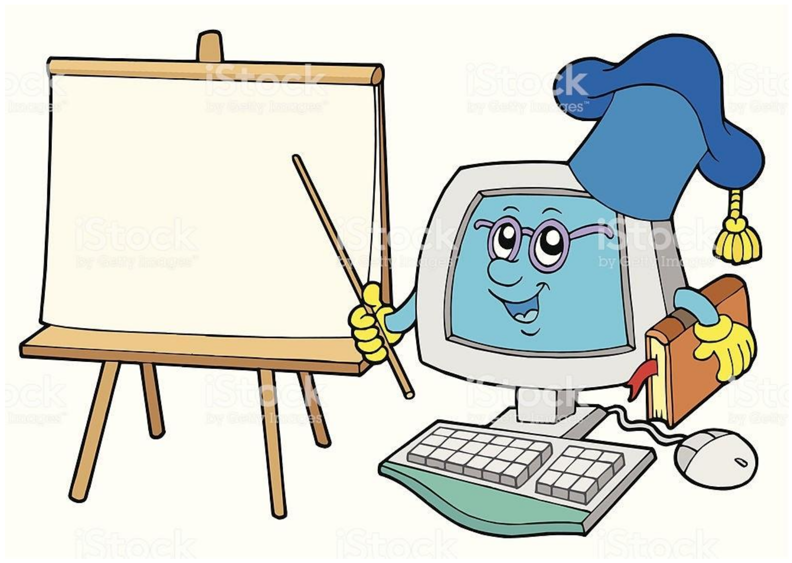
foundation of all

we do in life.

It shapes who we are

and what we

aspire to be.



J. KAVITHA, B.Sc, B.Ed, M.C.A, M.Phil.,
Computer Instructor Gr - I
GHSS, S.S.KULAM
Coimbatore – 641107.