Unit II
Structuring the Data, Computations and Program

Elementary Data Types : Primitive data Types, Character String types, User Defined Ordinal Types, Array types, Associative Arrays, Record Types, Union Types, Pointer and reference Type.
Expression and Assignment Statements: Arithmetic expression, Overloaded Operators, Type conversions, Relational and Boolean Expressions, Short Circuit Evaluation, Assignment Statements, Mixed mode Assignment.
Statement level Control Statements: Selection Statements, Iterative Statements, Unconditional Branching.
Subprograms: Fundamentals of Sub Programs, Design Issues for Subprograms, Local referencing Environments, Parameter passing methods.
Abstract Data Types and Encapsulation Construct: Design issues for Abstraction, Parameterized Abstract Data types, Encapsulation Constructs, Naming Encapsulations.

Subprograms

Subprograms:

Definition:

A collection of statements which can be reused to save memory space and coding time is known as a subprogram.

Difference Between Function and Procedure.

Function:

- The function is one of the fundamental thoughts in computer programming. It is used to calculate something from a given input.
- Hence it got its name from Mathematics. The function can be either user-defined or predefined. The function program has a block of code that performs some specific tasks or functions.

Procedure:

- In programming a particular set of instructions or commands are known as a procedure. Counting on the programming language it is known as a procedure, subroutine, function, or subprogram.

Example:

Function

```
int add(int x, int y)
{
int sum;
sum=x+y;
return sum;
}

void main()
{
int a;
a=4+add(3,5)+7;
}
```

Procedure

```
void add(int x, int y)
{
int sum;
sum=x+y;

}
```

Advantages of Subprograms:

a. Readability

b. Abstraction

c. Code reusability

Parameters	Function	Procedure
Basics	Functions calculate the results of a program on the basis of the given input.	Procedures perform certain tasks in a particular order on the basis of the given inputs.
Return	A function would return the returning value/control to the code or calling function.	A procedure, on the other hand, would return the control, but would not return any value to the calling function or the code.
Call	A function can be called using a procedure.	A procedure cannot be called using any function.
Compilation	The compilation of a function occurs when we call them in a program.	The compilation of the procedures needs to occur once, and in case it is necessary, these can be called repeatedly, and we don't have to compile them every single time.
Expression	A function must deal with expressions.	A procedure need not deal with expressions.

1. Fundamentals of Subprograms

Characteristics of Subprograms:

a. Each subprogram has a single entry point.

b. The calling program unit is suspended during the execution of the called subprogram, which implies that there is only one subprogram in execution at any given time.

c. Control always returns to the caller when the subprogram execution terminates.

Terminologies:

1. Subprogram Call

A subprogram call is the explicit request that a specific subprogram be executed.

2. Active Subprogram

A subprogram is said to be active if, after having been called, it has begun execution but has not yet completed that execution.

3. Subprogram Header

A subprogram header, which is the first part of the definition, serves several purposes.

e.g.

def adder parameters :

This is the header of a Python subprogram named adder .

Ruby subprogram headers also begin with def .

The header of a JavaScript subprogram begins with function.

In C, the header of a function named adder might be as follows:

void adder (parameters)

4. Body of Subprogram

The body of subprograms defines its actions.

In the C-based languages (and some others—for example, JavaScript) the body of a subprogram is delimited by braces.

In Ruby, an end statement terminates the body of a subprogram.

As with compound statements, the statements in the body of a Python function must be indented and the end of the body is indicated by the first statement that is not indented.

5. Parameter Profile

The parameter profile of a subprogram contains the number, order, and types of its formal parameters.

6. Protocol

The protocol of a subprogram is its parameter profile plus, if it is a function, its return type.

Unit II
Structuring the Data, Computations and Program

7. Subprogram Declaration

Subprogram declarations provide the subprogram's protocol but do not include their bodies. Function declarations are common in C and C++ programs, where they are called prototypes. In most other languages (other than C and C++), subprograms do not need declarations, because there is no requirement that subprograms be defined before they are called.

8. Formal Parameters

The parameters in the subprogram header are called formal parameters.

9. Actual Parameters

Subprogram call statements must include the name of the subprogram and a list of parameters to be bound to the formal parameters of the subprogram. These parameters are called actual parameters.

10. Positional Parameters

The correspondence between actual and formal parameters or the binding of actual parameters to formal parameters is done by position. Such parameters are called positional parameters.

11. Keyword Parameters

When lists are long, however, it is easy for a programmer to make mistakes in the order of actual parameters in the list. One solution to this problem is to provide keyword parameters, in which the name of the formal parameter to which an actual parameter is to be bound is specified with the actual parameter in a call.

e.g. Python functions can be called using this technique, as in

sumer(length = my_length, list = my_array, sum = my_sum)

where the definition of sumer has the formal parameters length, list, and sum.

Some authors call actual parameters arguments and formal parameters just parameters.

2. Design Issues for Subprograms

Following are the Design Issues for Subprograms:

1. Are local variables statically or dynamically allocated?

2. Can subprogram definitions appear in other subprogram definitions?

3. What parameter-passing method or methods are used?

4. Are the types of the actual parameters checked against the types of the formal parameters?

5. If subprograms can be passed as parameters and subprograms can be nested, what is the referencing environment of a passed subprogram?

6. Can subprograms be overloaded?

7. Can subprograms be generic?

8. If the language allows nested subprograms, are closures supported?

A closure is a nested subprogram and its referencing environment, which together allow the subprogram to be called from anywhere in a program.

3. Local Referencing Environments

a. Nested Subprogram

e.g. Javascript

b. External / Global Variable

c. Local Variable

- Stack Dynamic Local Variable

Advantages:

i. Flexibility

ii. Supports Recursion

iii. Shared Memory

Disadvantages:

i. Time to allocate and deallocate

ii. Indirect Access

- Static Local Variable

Advantages:

i. No time wastage for allocation and deallocation

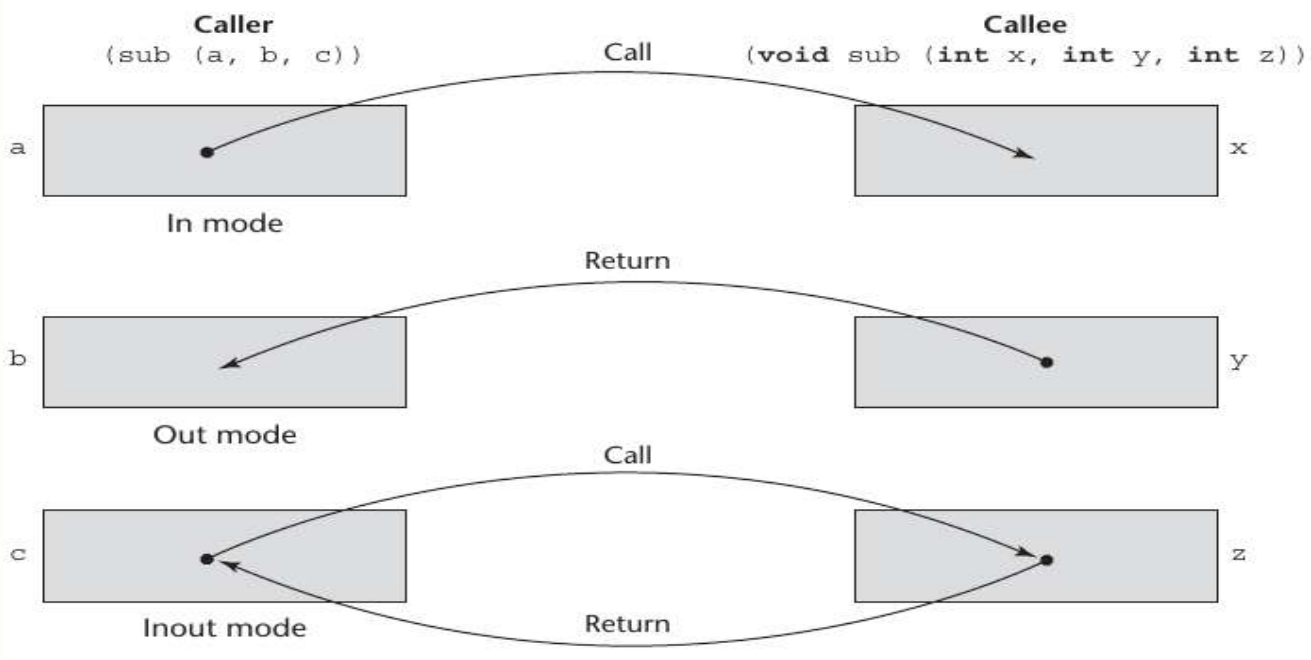
ii. Direct Access

Disadvantages:

i. Do not support recursion

ii. Storage cannot be shared

4. Parameter passing methods.



i. Pass By Value / Call By Value

When a parameter is passed by value, the value of the actual parameter is used to initialize the corresponding formal parameter, which then acts as a local variable in the subprogram, thus implementing **in-mode** semantics.

ii. Pass By Result

Pass-by-result is an implementation model for **out-mode** parameters.

e.g. C# code

```
void Fixer(out int x, out int y) {

x = 17;

y = 35;

}

. . .

f.Fixer(out a, out a);
```

iii. Pass By Value-Result

Pass-by-value-result is an implementation model for **inout-mode** parameters in which actual values are copied.

iv. Pass By Reference

Pass-by-reference is a second implementation model for **inout-mode** parameters.

v. Pass By Name

Pass-by-name is an **inout-mode** parameter transmission method.

When parameters are passed by name, the actual parameter is, in effect, textually substituted for the corresponding formal parameter in all its occurrences in the subprogram.