Essay Questions with Answers

3.1 SUB-PROGRAMMER

Q1. Define subprogram. Explain the characteristics of subprogram.

Answer :

A subprogram refers to the complex structure in programming languages that can perform some specific task for the main program. It is actually an abstraction operation in which a programmer evaluates the sequence of actions by considering them as a single action and controls the interaction of these actions with the other parts of a program.

Subprogram specification must specify the following information.

(i) The name of the subprogram.

(ii) The subprogram's signature consisting of the number of arguments, type of arguments, order of arguments and the return type of a subprogram.

(iii) The functioning of the subprogram must also be specified.

Characteristics

All subprograms must possess the following characteristics,

1. Only one entry point is allowed for each subprogram.
2. Only one subprogram must be under execution at any time.
3. Control is always transferred to the caller upon the completion of subprogram execution.

Q2. Discuss about generic subprogram in Ada.

Answer :

Generic Subprograms in Ada

A subprogram is said to be 'generic' if it takes different types of parameters on different activations or calls. It is also known as polymorphic-subprogram. The type of polymorphism exhibited by the overloaded Subprograms is called adhoc polymorphism.

The other type of polymorphism supported by a subprogram is called the parametric polymorphism, in which a subprogram takes the parameter of generic type. Ada and C++ supports compile-time parametric type of polymorphism.

Example

```
generic
    type Array_Index is(< >);
    type Array_Element is private;
    type vector is array(Integer range < >) of Array_ Element;
    procedure GENERIC_SORT(Lst in out vector);
    procedure GENERIC_SORT(Lst: in out vector) is
        Temp_Var: Array _Element;
    begin
        for Elem_Top in Lst
        'First.. ..Arrayjndex'predecessor (Lst' Last) loop
            for Elem_Bottom in
        Array _Index' successor(Elem_Top)....
```

```
        Lst' Last loop
        if Lst(Elem_Top) > Lst(Elem_Bottom)then
        Temp_Var := Lst(Elem_Top);
        Lst(Elem_Top) Lst(Elem_Bottom);
        Lst(Elem_Bottom) := Temp_Var;
        end if;
        end loop;
    end loop;
end GENERICJSORT;
```

Q3. What are coroutines?

Answer :

Coroutines

It refers to a special kind of subprogram that contains multiple entry points which are under the control of other coroutines. This control mechanism is called the symmetric unit control model. Coroutines also have some means of keeping the status information between the activations. Hence, they are history-sensitive with static local variables. The coroutines invocation is called a **resume** but not a call as it starts its secondary execution from a point it has terminated rather than from its beginning.

One of the most general characteristics of subprograms is **Quasi-concurrency.**

Coroutines in an application are generated by a program called the **master unit,** which is not actually a coroutine. When the coroutines are created by a master unit, they execute their initialization codes and return the control back to their master unit after their completion.

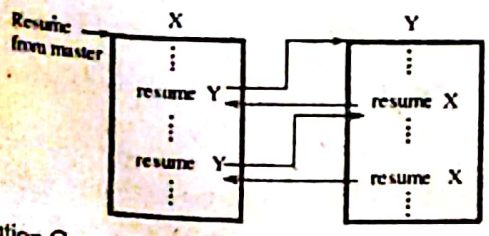
Example

Consider a program code which consists of two coroutines *X* and *Y*. The sequence of execution for two coroutines is as follows,

(i) The master unit starts the execution of coroutine *X*.

(ii) After completing the execution of some code, *X* initiates the execution of *Y*.

(iii) When coroutine *Y* returns the control to coroutine *X*, it's execution is resumed from the point it has previously suspended.



Execution Control Sequences for the Two Coroutines

3.2 Design Issuer of Subprogram

Q4. Discuss the design issues of subprogram.

Answer :

Design Issues

(i) What method(s) is/are used for passing the parameters to a subprogram?

(ii) Are the local variables, statically or dynamically allocated?

(iii) Can the definitions of subprograms be nested within one another?

(iv) Is it possible to overload subprograms?

(v) Are the subprograms generic?

(vi) What is the referencing environment for the nested subprograms that are passed as parameters?

(vii) Does any type-checking takes place between the actual parameters and the formal parameters?

(viii) Is separate/independent compilation supported?

Q5. Discuss design issues for functions.

Answer :

Design Issues for Functions

1. Are the side effects allowed by a function?
2. What are the types of values returned by a function?

1. Side Effects of a Function

In order to avoid the problems that arise due to the side effects of the called function, the parameters to a function must be in mode parameters. It is needed by some of programming languages such as 'Ada', which allows only in-mode specification for the formal parameters.

2. Types of the Returned Values

The types of the returned values are restricted in most of the imperative programming languages. A function if C can return the value of any type except arrays and functions, because a return value of an array or function type can be handled by pointers.

A function in C++ can return a value of user-defined types or classes a pent from returning values of types allowed in C.

3.3 Local Reference Environment

Q6. Explain the local reference environment.

Answer :

It can be defined inside a subprogram, whose scope is confined to the subprogram. Defining local variable creates its local referencing environment. The variables enclosed within a subprogram are called 'Local Variables'.

Local variables are of two types,

1. Static variables
2. Stack-dynamic local variables.

1. Static Variables

The variables whose declaration involves the use of 'static' keyword. The lifetime of these variables is throughout the program execution.

Advantages

(i) More efficient. This efficiency results from 'direct addressing'.

(ii) History-sensitive

(iii) Don't required run-time allocation and deallocation.

(iv) No support for indirection is provided the static variables.

Disadvantages

(i) They do not support recursion.

(ii) Storage space occupied by the static local variables.

2. Stack-dynamic Local Variables

These are the local variables which are bounded to the storage at the beginning of subprogram, execution and are unbounded upon the termination of subprogram.

3.4 Parameter Passing

Q7. Explain in detail about parameters.

Answer :

Parameters

Parameter is defined as a type of variable used in subprograms in order to access a part of data, which is provided as input to the subprogram. Basically, two approaches non-method subprogram so as to access the data that is to be processed:

1. Direct access method
2. Parameter passing method.

1. Direct Access Method :

It is made to nonlocal variables that are declared somewhere in the program but are visible within the subprogram.

2. Parameter Passing Method

It is more flexible parameterized computation is performed where in data is passed as parameters. Thus data is then accessed via names that are local to the subprograms.

Types of Parameters

The different types

1. Formal Parameters

Which are present in function definition or function prototype. The formal parame are also called as dummy arguments or parametric variables. Whenever a for is called the values of actual parameters are copied to formal parameters.

2. Actual Parameters

The paramters of a calling subprogram are called actual parameter.

```
#include<stdio.h>
#include<conio.h>
int sum(int, int);
main( )
{
    int x,y, sum;
    printf("Enter two integers");
    scanf("%d%d", &x, &y);
    sum = fun(x, y);
    printf(" sum = %d", sum);
}
int fun(int a, int b)
{
    int s;
    s = a + b;
    return s;
}
```

x, y are actual parameters

a, b are formed parameters

3.5 PARAMETER THAT ARE SUB PROGRAMM CALLING UNDIRECTLY

Q8. Discuss about the sematic models of parameter passing.

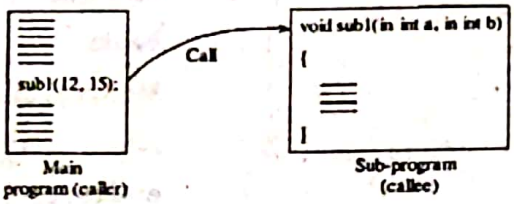
Answer :

There are three different semantic models of parameter passing.

1. **In mode**

The formal parameters receive data from corresponding actual parameters.

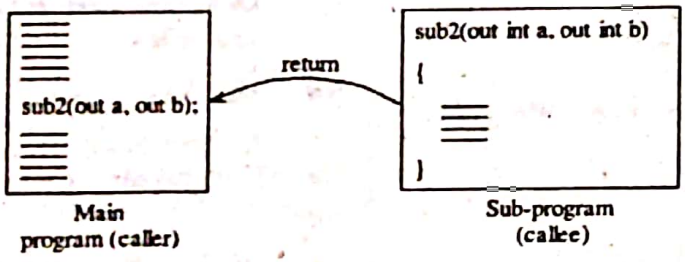
Example :



2. **Out mode**

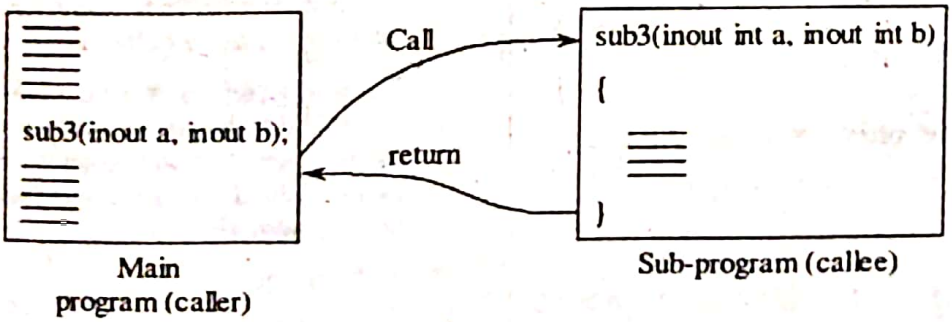
The formal parameters send data to acutal parameters.

Data transfer is from caller to catter.



3. **Inout mode**

This mode is a combination of both inmode and out mode. It provides two way data transfer i.e., a caller can send parameters to a calle and intern calle can send result to the caller via the same parameters.



3.6 OVERLOAD SUBPROGRA

Q9. Write about overloaded subprograms.

Answer :

Subprogram overloading is a process of using the same (function) name for many subprograms. The overloaded subprograms may differ in the number, type or order of its parameters. Java, C++, Ada and C# provides the feature of subprograms overloading.

When an overloaded subprogram is called, Java or C++ compiler first checks the subprogram name and then the number and type of parameters are analyzed to decide which version of the overloaded subprogram must be chosen for execution. This process is also known as 'polymorphism'.

Example

```
class Rectangle
{
    double len, breadth;
    Rectangle(int x)
    {
        len = breadth = x;
    }
    Rectangle(int x, int y)
    {
        len = x;
        breadth = y;
    }
    int area()
    {
        return(len * breadth);
    }
}
class Example
{
    public static void main(String args[ ])
    {
        int a1, a2;
        Rectangle r = new Rectangle( 10,20);
        a1 = r.area( );
        System.out.println("Area =" + a1);
        Rectangle s = new Rectangle(10);
        a2 = s.area();
        System.out.println("Area = " + a2);
        }
}
```

3.7 User defined Over loaded Operators

Q10. Discuss user - defined over loaded operators.

Answer :

User-defined Overloaded Operators

Ada and C++ allows operators to be overloaded by the user.

Consider the following function in Ada that overloads the multiplication operator for determining the dot product of two arrays having the same length. The dot product of the equilength arrays is the sum of the products of the corresponding elements in the two arrays.

Example

```
Let vector be an integer type array
function "*" (X,Y: in)
return integer is
sum: Integer: = 0;
begin
for Arr_Index in X' range loop
sum: = sum + X (Arrjndex) * Y (Arr_Index);
end loop;
return sum;
end "*";
```

3.8 ADT

Q11. Describe abstract data types with suitable examples.

Answer :

Abstract data types are the user-defined data types that exhibits the same characteristics as predefined data types.

An ADT refers to a data type that must satisfy the following two requirements.

1. A single syntactic unit must contain the type declarations and the operations that can be performed on objects of the declared type which gives the types interface. The implementation of the type along with its operations can be written either in the same or the different syntactic unit.
2. The objects representation (of the declared type) is completely hidden from the program unit that uses the type. So, the direct operations that can be performed on the objects are those specified in the types definition.

Example languages :

Simula 67, ADA, Modula-2,

C++ Java.

Features of ADT

An ADT supports three features.

1. Abstraction
2. Encapsulation
3. Information hiding.

1. **Abstraction**

It is a process of showing relevant details and hiding irrelevant details from the user i.e., the implementation details are completely hidden from the user.

2. **Encapsulation**

Encapsulation is a process of binding together, both the code and data in such a way that it can be prevented from any interference and misuse.

3. **Information Hiding**

The data members of an ADT can't accessed outside the unit but instead they can be accessed by using the member functions of an ADT.

Q12. Discuss design issues of ADT.

Answer :

Design Issues

1. The abstract data types must be parameterized.
2. It should provide a syntactic unit which will be used for encapsulating the type and subprogram definitions of the abstract operations.
3. The type name and subprogram headers must be written in such a way that they must be visible to the clients of abstractions.
4. The type name and subprogram headers must be visible and the type definitions and subprogram body must be hidden.
5. The objects in ADTs must be provided with built-in operations.
6. The designer of ADTs must provide some frequently used operations.

Q13. Explain parameterized ADT in C++.

Answer :

C++ not only supports generic ADT's but also supports parametrized ADT's . Let us, consider a stack program in which the element type of a stack are made generic. This is done by defining stack class as a template class. The class definition can include two constructors, thereby enabling the users to use either variable integer size, or fixed integer size of stack.

```
#include<iostream.h>
template<class stk>
class Stack
{
    private:
    stk *stackpointer;
    int max;
    int topptr;
    public:
    Stack( )
    {
    stackpointer = new stk[50],
    max=49,
    topptr = -1;
    }
    Stack(int size)
    {
    stackpointer = new stk[size];
    max = size-1;
    topptr = -1;
    }
    ~Stack( )
    {
        delete stackpointer;
    };
    void push(stk num)
    {
    if(topptr = = max)
    cout<<"stack is full \n";
    else
    stackpointer [++topptr] = num;
    }
    void pop ()
    {
    if(topptr = = -1)
    cout <<"Stack is empty \n";
    else
    topptr - - ;
    }
    stk top( )
    {
    return (stackpointer[topptr]);
    }
    int empty( )
    {
    return (topptr = = -1);
    }
}
```

Q14. Explain about the naming encapsulation?

Answer :

1. Large program define many globla names nied away to divide into logical groupings.
2. A naming encapsulation is used to create a new scope for names.

C++ Names Spaces

1. Can place each library in its own name space ans quality names used outside with the name space.
2. C# also includes name spaces.

Java Packages

1. Packages can contain more than one class definition, classes in a package are partial friends.
2. Clients of a package can use fully qualified name or use the import declaration.

Ada Package

1. Packages are defined in hierarchial which corresponds to file.
2. Visibility from a program unit is gained with the clause.

Ruby Modules

1. These are name encapsulation but it has modules.
2. Modules can't be instiabtiated or subclass & they can't by define variable.

Q15. Discuss about the encapsulation constructs.

Answer :

Large programs have two special

1. Some means of organization other than simply division into subprogram.
2. Some means of partial compilation obvious solution a grouping related into a unit that can be separately compiled such collections called encapsulation.

Encapsulation in C

1. Files containing one or more sub programs can be independently compiled.
2. The interface is placed in a header file.
3. The linker doesnot check types between a header and associated implementation.
4. # include preprocessor specification

Encapsulation in C++

1. Can define header and code files.
2. Classes can be used for encapsulation
3. Friends provides a way to grant access to private members of a class.

Ada Packages

1. Ada specification packages can include any no. of data and subprogram declaration.
2. Ada can be compiled separately.
3. A packages specification and body parts can be compiled separately.

3.9 COMPUTING SUB-PORGRAM

Q16. Explain about the dynamic scoping.

Answer :

Dynamic Scope

The process of determining the scope of a variable at run time is called 'Dynamic scope'.

It does not depend on the spatial relationship between the subprograms instead it depends on the calling sequence of the subprograms.

When dynamic scope rules are applied to the non-local references in the previous example, the variable y referenced in subprg_1 is dynamic and hence can't be determined at compile time. Depending on the calling sequence, a reference to the variable y can be made from either of the two declarations of y.

The accurate meaning of y can be determined at run time by starting the search procedure with the local variables declarations of the dynamic-parent or a calling procedure are searched, even then, if no declaration for y is found, then the search procedure continues till its declaration is found in any of the dynamic parents.

Q17. Discuss about the blocks.

Answer :

Block

A block refers to the section of code which is grouped together block consists of one or more declaration and statements.

A programm language that permits the creation of blocks including blocks nested within another blocks is called block structured programming language.

The notation R a block is introduced by different syntax in different language. In Algol family in which the blocks one delimited by the keywords begin and ends.

In C, C++, Java family in which the block are delimited by curly braces.

Ex :

```
if (a[i] M<a[j])
{
int t;
t = a[j];
a [j] = t;
}
```

Variable 't' is declared inside the if block hence the scope is restricted to if block. Thus scope of the variable depends on the block. This is due to the fact that the local variable are confined to the block in which they are declared.

Q18. Explain about general stemantics of calls and returns.

Answer :

The subprogram call and return operations of a language are together called its subprogram linkage.

General semantics of calls to a subprogram responsible for

1. Parameter passing methods for in and inout arguments.
2. Stack vs dynamic allocation of local variables.
3. Same the execution status of calling program.
4. Transfer of control and arrange for the return.
5. If subprogram nesting is supported, access to nonlocal variables must be arranged.

General Kinematics of sub program returns responsible for

1. Out mode and inout mode parameters must have their values returned.
2. Deallocation of stack or dynamic locals.
3. Resotre the execution status.
4. Return control to the caller.

Q19. Discuss with nested subprograms.

Answer :

Some non C-based static -scoped languages FORTRON 95+, Ada, python, Java script, Ruby and (va) u stack-dynamic local variables and allow subprogram to be nested.

1. All variables that can be non locally accessed reside in some activation record instance in the stack.
2. The process of locating a non local reference.
   - a) Find the correct activation record instance.
   - b) Determine the correct of set within that activation record instance.

Locating a non-local Reference

1. Finding the offset is easy
2. Finding the correct activation record instance.

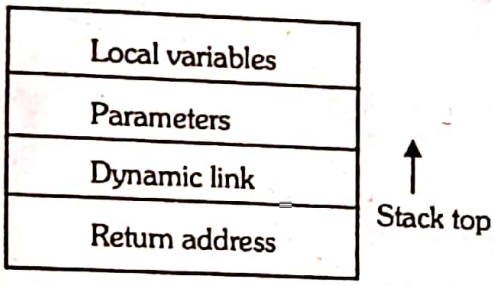
Q20. How implement subprograms with stack-dynamic local variables? Explain?

Answer :

How complex activation record.

1. The compiler must generate code to cause implicit allocation and deallocation of local variables.
2. Recursion must be supported.

Typical activation record for a language with stack - dynamic local variables.



3. The activation record format is static, but its size may be dynamic.
4. The dynamic link points to the top of instance of the activation record of the caller.
5. An activation record instances reside on run - time stack.