9.4

The characteristic that makes Python subprogram different from subprogram of other languages is as follows:

The def statements in Python are executable statements. The function body is assigned the specified name when the def statement is executed. Until the def statement of the function is executed, it cannot be called.

Consider an example,

if

def sub():

...

else

def sub():

...

Here the function sub() cannot be called until its def statement is executed. Only that version of the function can be called whose def statement is executed. In the above example either the than clause or the else clause gets executed. If the than clause gets executed, than the version of function defined here can be called, and the version of the function that is defined in the else clause cannot be called.

9.10

Procedures and functions differences

Subprograms are of two types- procedures and functions. Both procedures and functions are block of statements with some name associated with this block of statements. Functions return values to the caller while procedures do not return any value to the caller.

Some programming languages use functions as procedures by allowing functions to not to return values. For example, C uses void keyword to indicate function do not return a value to the caller function.

There is some difference in the semantics of procedures and functions.

Procedures are used to define new statements in a programming language while functions resemble mathematical functions. In FORTRAN, procedures are called as subroutines.

Functions can be used in an expression while procedures cannot be used in the expressions. For example,

int power(int a, int b)

{

int c=a;

while(b>1)

{

b--;

a = a\*c;

}

}

void main()

{

int x=5;

x= x + power(5,3);

}

Here the function call is used in the expression. The value returned by the function power () is added to the variable x.

9.11

Design issues of subprogram

Subprograms are group of statements with some name associated with this group of statements. Adding the concept of subprograms to a programming language requires the compiler to perform some extra functions.

There are several design issues with the subprogram. Those are

1.) Memory to be allocated to variable can be either static or dynamic. When designing a subprogram, an issue of in what way memory should be allocated arises.

2.) Which parameter passing method should be used in the programming language?

3.) Should nesting of subprogram be allowed in the programming language?

4.) Type checking between actual and formal parameter are checked or not?

5.) Overloading of subprogram should be allowed in the programming language or not?

6.) Should programming language be allowed to define generic subprograms or not?

7.) If nested subprograms are allowed in a programming language, should closures (type of subprograms that also include referencing environment) be included or not?

8.) If subprograms can be nested and can be passed as parameters in other subprogram call then what should be the referencing environment of the passed subprogram?

9.13

Static local variables

Static local variables are the variables which are allocated memory when the subprogram is compiled.

Advantages:

The advantages of stack dynamic local variable are as follows:

(1) The overhead of run time allocation and de-allocation is not required.

(2) Static local variables can be accessed directly or indirectly. Direct access to variables is more efficient than the indirect accesses.

(3) Static variables retain data values over function calls.

Disadvantages:

The disadvantages of static local variables are as follows:

(1) The programming languages that allow only static variable do not support recursion.

(2) The memory allocated to static variables cannot be shared between variables of different subprograms.

9.15

Parameter passing

When we are pass parameters to and from the subprogram, it is called the parameter passing methods. Different semantics models of parameter passing are as follows:

(1) In mode

(2) Out mode

(3) Inout mode

In mode:

The formal parameters which are defined in this model can receive data from the corresponding actual parameters.

Out mode:

These formal parameters can transfer data to the corresponding actual parameters.

Inout mode:

These formal parameter can transfer data to and from corresponding formal parameters.

Example:

Consider an example in which variable p, q and r are defined in in mode, out mode, inout mode respectively.

9.30

Design Issues of functions

Functions are the group of statement with some name associated with it. When function completes its execution it returns a value to the calling function or subprogram. There are several design issues with functions. These are as follows:

(1) Functional side effect is allowed or not.

(2) Types of values are returned by the function.

(3) Number of values returned by the function.

Functional side effect

Functional side effect arises if the called function changes the value of actual parameter or any of the global variables. To avoid functional side effect parameters should always be in-mode format only.

In-mode parameters restrict the called subprogram to have access to the actual parameters. Pass by reference method can be used to allow called function to have access to the actual parameters.

Example:

Consider a C program

void fun(int b)

{

b=30;

printf("the value of b=%d",b);

}

void main()

{

int a=20,x;

x= a + fun(a);

}

Here the variable “a” ...actual parameter a, which avoids the functional side effect.

Types of returned values

Functions always return a value and it is the programming language which decides what types of values are returned by the function. C allows its functions to return any type of value except arrays and functions. Pointers to arrays and functions can be returned by a function. Functions of Python, Ruby and Lua can return values of any type.

Number of returned values

Languages are designed to allow functions to return values to the calling subprogram. Most of the languages allow their functions to return only a single value. Functions in ruby, lua can return multiple values. If multiple values are returned then there must be multiple acceptors in the calling subprogram.

Example:

return 3, sum, difference; // function return statement

a, b, c=fun(); // function call statement

Here three values are returned by the function. These three values are accepted by the variable a, b and c respectively.

11.5

Language Design Issues for Abstract Data types

The language design issues for abstract data types are :

1. The first design issues is the form of the container for the interface to the type. Here, issues trying to say the data of a single type or more than one type.

2. The second design issues are whether abstract data types can be parameterized. A parameterized abstract data type means that the data type is generic. These generic types are considered templates.

3. The third design issues are how access controls are provided and how such controls are specified. Here, access control means visibility of objects.

Example is simula67 language provided encapsulation but with-out information hiding.

In language packages use for encapsulation with information hiding.

Example in :

package … is

type NODE\_TYPE is private;

…

private

type NODE\_TYPE is

record

…

end record;

11.19

Reasion for Java does not have Destructors

Java does not have destructors because Java has already integral memory handling mechanism that is also known as garbage collection , the main function of which is to clear off the unused memory automatically.

The garbage collection process in Java automatically frees the memory when it is no longer referenced by the program. This makes it simpler for programmer as he does not have to manually deallocate the memory, thus reducing bugs.We already know that the main function of the destructor is to clear off unused variables and clean up the memory, Hence there is no requirement for destructor.

11.23

Friend function:

• Friend function is a function that is not a member of a class but has access to class private and protected member function.

• It acts as a friend to a class. But it must be listed in the class definition.

• It is a global function that can access all data members of a class. Friend functions are always defined from outside the class without using the name of the class and the scope resolution operator (::).

• friend keyword is used to make any function friend, then we can use the private data of a class with the help of an object.

Example :

class class\_name

{

friend return\_type fun\_name(arguments)

private :

…

public :

…

};

return\_type fun\_name(arguments) {operation… }

int main()

{

…

}

Friend class:

• A friend class can access the private and protected member of the class in which it is declared.

• All member functions of the friend class become a friend of the class in which the friend class was declared. Friend status is granted by using ‘friend’ keyword.

• It is one which has been declared inside another class to make it access the private member of the class which has extended its friendship.

Example :

class A

{

private :

…

public :

…

friend class B;

};

class B

{

…

}

11.26

Struct have ability to create types with value semantics, which, if properly used, leads to better performance in a managed environment. In C#, structs are value types. There are two ways to create value types in C# using the enum keyword and the struct keyword.

Example of struct object creation using struct keyword :

class program

{

struct S

{

public int x;

public int y;

};

static void main()

{

S local;

local.x = 1;

local.y = 2;

}

}

Here we make a struct object called ‘S’ having x and y as data members and its object as ‘local’.

11.31

Instances Template Classes

Templates are a way of making classes without knowing what data type will be handled by the operations of the class. Templates define the behavior of the class. It is known as generic programming. A template class focuses more on algorithm pattern rather than the specific data type. Due to template class, it makes classes more reusable.

If we are not using the template class concepts, then we have to make more than one stack for different data type, but in the case of template classes, only one stack declaration is fine because the template class does not depend upon the return data type. We can make our own stack data type. It can handle a stack of any data type.

The basic syntax for a template class:

template class a\_class

{

…

}

Here, a\_type is not a keyword. It is an identifier that will represent a single data type.

Defining variable in the class: a\_type a\_var;

To declare an instance of a template class, the syntax is:

a\_class <int> an\_example\_class;

12.3

Multiple Inheritance

The ability of a class to inherit the properties and methods of more than one class is known as multiple inheritance. In simple words it can be said that a child class can have more than one parent class. The figure shown in below:

Multiple Inheritance

Class D inherits properties from class A, B and C. Thus A, B, C are the multiple parents of the child class D.

12.4

Polymorphic Variable

A variable that can refer to different types of data during the time of execution is known as the polymorphic variable. The program will decide which form of the data to use at the time of execution.

Generally it is used to refer different types of objects. A variable of parent data type can refer any object of the child class. When a polymorphic variable is used as a parameter, it is known as the pure polymorphism.

Example:

/\*The following example gives a skeleton of a program to illustrate the concept of polymorphic variables\*/

class Parent ()

{

public:

/\*…\*/

};

class child (): public Parent

{

/\*…\*/

};

void main ()

{

parent p;

//polymorphic variable

p=new child ();

/\*…\*/

}

Here a variable of class Parent is referring variable of class child.

12.7

Multiple inheritances are preferred when we need some common behavior from two distinct classes. But interfaces just carry method signature whereas classes carry actual behavior. Multiple inheritances greatly reduce boilerplate code.

Example:

Interface hold only method signature.

class AbstractClass

{

public:

virtual void display() = 0;

// Pure virtual function makes this

//class Abstract class.

// Virtual function.

virtual void search();

void getval();

};

class student

{

public:

int rollno;

char name[10];

char subject[10];

void getdetail()

{

...

...

}

};

class marks

{

public:

int m1;

int m2;

int m3;

void getmarks()

{

...

...

}

};

class student :public student, mark

{

void showdetail()

{

getdetail()//functionality is passed

...

}

void showmarks()

{

getmarks()//functionality is passed

...

}

};

12.10

Nesting Class

Java allows a class to be defined within another class. The class that is defined inside another class is called a nested class. Nested classes are of two types.

1. Static nested class and

2. Non-Static nested class

The nested class has the following structure:

class outerclass

{

…

class nestedclass

{

//class is defined inside another class

…

}

}

1. Static nested class: Static nested classes are declared with a keyword ‘Static’. They can’t access other members of the enclosing class. Static nested classes can be public, private or protected.

2. Non-Static nested class: Non-Static nested classes are also known as inner class. They can access members of the enclosing class. They can also access private members of the enclosing class.

Example: This example gives the skeleton of Static and Non-Static nested class.

class outclass ()

{

…

static class staticnested ()

{

…

}

class innerclass ()

{

…

}

}

Advantages: The following are the advantages of nesting classes:

1. Logically groups classes

2. More readable

3. It makes code maintenance easy and

4. Increases Encapsulation

12.16

Influence of SMALLTALK on computing environment

1. It was the model for message passing type of computation.

2. Its WIMP GUI inspired the today’s windowing environment for Personnel computers.

3. It first gave the idea of IDE (integrated development environment).

This idea was observed from SMALLTALK’s code browser and debugger.

Haskell program to determine the GCD of two positive integer numbers

