if i=0 & j>0 this construct choose non-deterministically blw the 1st and 3rd assign-ment. if i=j and is not zero a run time error occur bcz none of the condition is true.

PPL Unit-3 S-2

Subprogram

Subprograms are the fundamental building block of a program. It is the imp concept in programming language design. The reuse result in different kind of siblings include memory space and coding time.

A subprogram is a description of actions of the program abstraction - A subprogram called is an explicit request that the called program be executed.

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

A subprogram header is the 1st line of the definition that specify the syntactic unit and provide a name for the subprogram and optionally specify a list of parameters.

Subprogram
In 'C'
void adder(Parameters)
In Ada
procedure Adder(Parameters)
In FORTRAN
Subroutine Adder(Parameter)

Spiral

Spiral

Date.....

Date.....

* Parameters

They are of two types -
(i) Actual
(ii) Formal

A sub program declaration provide the protocols but not the body of the subprogram. A sub-program used the formal parameter as dummy variable as found in the function definition while actual parameter represent a value or address used in the subprogram called statements.

Function declaration are common in C, C++ programs where they are called prototypes. Subprogram typically describe the combination and there are two ways that the non-local method program can gain access to the data that is to the process.

(i) Through the direct access to non-local variables

The only way the computation can proceed through different data and assign the new value to those non-local variables b/w calls to the subprogram.

(ii) Through the parameter passing

Spiral

Spiral

Date.....

Data pass Through the parameter that are access through the names that are local to the subprogram.

* Design issues for subprograms

(i) Parameter passing
(ii) Type checking of parameters
(iii) Static or Dynamic
(iv) Nesting
(v) Referencing environment
(vi) Overloading

* Functions and Procedures

There are 2 different categories of subprograms as procedure and functions

(i) Procedure

It provide user define parameters computation statements - The computation is done by a single call statement.
Procedure can produce result in the calling program by two methods.
→ If there are variables that are not formal parameters but are still visible in both the procedures and the calling program unit. The procedure can change them.
→ If the subprogram has formal parameters that

Spiral

Date.....

allow the transfer of data to the caller and those parameters can be changed.

(ii) Functions

Functions define user-defined operators which are semantically modelled on mathematical functions - It modify neither its parameter or any variable define outside the function.

The method of java, C++, C# are syntactically similar to the functions of C

* Referencing Environment

The referencing environment of a statement is a collection of all the names that are visible in the statement and the collection of scope that are examined in order to find a binding.

(i) Local referencing environment

The set of association created on entry to a subprogram that represent a formal parameter. The variables that are defined inside the subprogram are called local variables. They can be either static or stack dynamic bound to storage when the program begin execution and unbound when execution terminates.

Spiral

Stack dynamic

In C function locals are ~~static~~ unless specifically declared to be static. For ex.

```
int adder(int list[], int listlen)
{
  static int sum = 0;
  int count;
  for (count = 0; count < listlen; count++)
    sum += list[count];
  return sum;
}
```

Note - Ada subprogram and the methods of C++, Java, and C# have only stack dynamic local variables.

(ii) Global referencing environment

If the association created at the start of execution of the main program, then these associations form the global referencing environment of that subprogram.

(iii) Non-local referencing environment

The set of association for identifier that may be used within a subprogram but that are not created on entry. It is termed as non-local referencing environment of the subprogram.

Spiral

Date

X Parameter Passing Methods

1. Semantic Models of Parameter passing

Formal parameters are characterised by one of the three semantic models -

(i) In mode

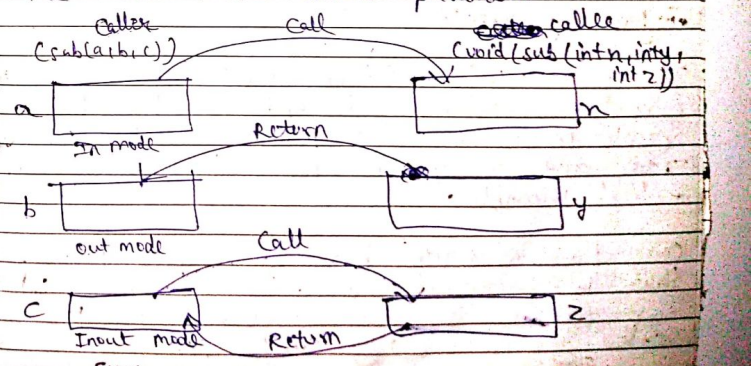
They can receive the data from the corresponding actual parameter

(ii) Out mode

They can transmit the data to the actual parameter.

(iii) Inout mode

They can do both transmit and receiving the data from actual parameters.



Spiral

Date.....

2- Implementation model of Parameter Passing

(i) Pass by value

When a parameter is passed by value the value of actual parameter is passed. ie. used to initialise the corresponding formal parameter which then act as a local variable in the subprogram. This is basically implementing the in-mode semantics.

Disadvantage of this method is an additional storage is required for the formal parameter. This in the old subprogram or some area outside both the caller and the called program. The actual parameter must be copied to the storage area for the corresponding formal parameter. If the parameter is large such as an array it would be costly.

(ii) Pass by result

It is an implementation model for out mode semantics - when a parameter is passed by result no value is transmitted to the subprogram and corresponding formal parameter acts as a local variable.

One problem with pass by result model is that there can be a actual parameter collision such as the one created with the call. For ex- sub(P1, P1)

Spiral

Date.....

In this two formal parameters that assign diff- value than whichever of the two is copied to their corresponding actual parameter last become the value of the P1.

(iii) Call by reference

It is second implementation for inout mode parameter rather than copy the data value back this method transmits an access path or we can say an address to the old subprogram. This provides the access path to the call storing the actual parameters. An actual parameter is shared with the called subprogram.

The advantage of this method is efficient in terms of time and space and duplicate space is not required.

(iv) Pass by name

```
procedure double(n);
    read n;
begin:
    n := n+2
    end;
```

This method is an in-out mode parameter transmission that does not correspond to a single implementation model. The symbolic

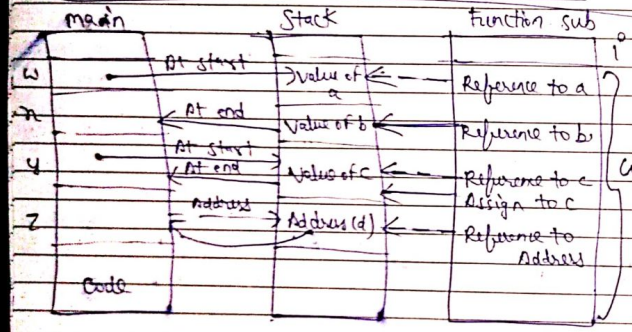
Spiral

Date.....

name of variable is passed and allowed to access and update.

When parameter are passed by name the actual parameter is textually substituted for the corresponding formal parameter in the subprogram.

* Implementing Parameter Passing Methods



The runtime stack is initialised and maintained by run time system which is a system program that manage execution of the program.

The runtime stack is used for subprogram control and parameter passing.

The subprogram sub is called from main with the call sub(w, n, y, z) where w is the pass by value, n is the pass by result,

Spiral

Date.....

and y is the passed by value result and z is the passed by reference.

The subprogram that is based on the parameters that address the place in the stack and access to the formal parameters in the old subprograms that is indirect the addressing from the stack location of the address.

* Parameters that are subprograms Names

In language that allow the nested subprogram such as JavaScript and these is another issue related to subprogram that are passed as parameters.

What will be the referencing environment for executing the passed subprogram that should be used as three choices.

(i) It is the environment of called statement that represent the passed subprogram as "shallow binding".

(ii) It is the environment of the definition of passed subprogram as "Deep binding"

(iii) It is the environment of call statement that pass the subprogram as an actual parameter that is "Adhoc binding" that has never been used.

* Overloaded subprograms

Spiral

Date

An overloaded subprogram is a subprogram that has the same name as another subprogram in the same referencing environment. Every version of overloaded subprogram must have a unique protocol that must be different from the others in no. order, or type of its parameter. or in its return if it is a function.

The meaning of a call to a overloaded subprogram is determined by the actual parameter list. Users are also allow to write multiple versions of subprogram with the same name is Ada, Java, C++, C#.

But overloaded subprogram that have default parameter as lead to ambiguous subprogram calls.

* Generic subprogram

It enable the programmer to write a general algorithm which will work with all the data types. For ex- template programming in C++.

The overloaded subprogram provide a particular kind of polymorphism called adhoc polymorphism.

Parametric polymorphism is a provided by a subprogram that takes a generic parameter that is used in a type expression that

Spiral

Date

describe the types of parameter of the subprogram.

The generic function in C++ have the descriptive name of template functions

```
#include<iostream>
# include<conio.h>
using namespace std;
template<class t, class t>
Void sum (t, a , t b)
{
cout<<"sum ="<<a+b <<endl;
}
main():
{ int a ,b;
 float x, y;
cout<<" Enter two integers :";
 cin>> a >> b;
cout<<"Enter two float no :";
 cin>> x >>y;
 sum(a,b);
 sum (x,y);
 cum(a,x);
}
```

* Type checking parameters

It is widely accepted that the reliability demand the type of actual parameters to be checked for consistency with the types of corresponding formal parameter. For ex.

Spiral

Date.....

Result = Sub1(1)

The actual parameter is integer constant if the formal parameter of Sub1 is a floating point type no error will be detected without parameter type checking. Such as early languages FORTRAN 77, the original version of C did not required the parameter type checking.

Later on Perl, PHP and Java scripting don't have type checking but Pascal, FORTRAN 90 and Ada language it is always required the type checking.

* Design issues for functions

(i) Side effect

The problem of side effect of functions that are called in expressions. The parameters to function should always be in mode. Some languages such as Ada language can have only in mode formal parameters. This effectively prevent a function from causing the side effect through its parameters or aliasing of parameters. In most other languages however the functions can have either pass by value or pass by reference parameters that allow the functions and may cause side effect and aliasing.

Spiral

Date.....

(ii) Types of Return value

Most imperative programming language restrict their to the types that can be returned by functions. FORTRAN 77, PASCAL and Modular 2 functions allow only unstructured type to be return

C allow any type to be return by its function except array. C++ is like C but also allow the user defined types or classes to be written from its functions. Java script functions can be passed as parameters and return from functions.

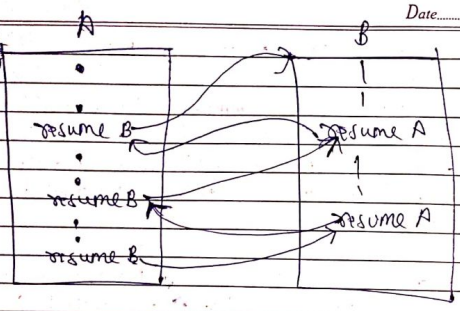
* Coroutines

A coroutines is like a subprogram that has multiple entities and control them itself it is also called symmetric control. A coroutine is a function that can suspend the execution and to be resumed later. It also maintained their status b/w activation.

The 1st resume of a coroutine is to its beginning but subsequent call enter at a point just after the last executed statement in the coroutine.

Spiral

Date.....



Suppose program units A and B are coroutine. They are two ways an execution sequence that involving A and B might proceed the execution of coroutine A is started by master unit after some execution A start B when coroutine B first control to return to coroutine A. The semantic is that coroutine A from where it ended its last execution. and local variables have the value ~~and global variables have the value~~ left then by the previous activation.

* General semantics of calls and return

The subprogram called and return operation of a language are called subprogram linkage. Call consist the following -)

(i) Mechanism for parameter passing.
(ii) Allocation and binding of local variables.
(iii) Save the execution status of calling unit.
(iv) Arrange for transfer of control and return

Spiral

Date.....

(v) Arrange for access to visible non-locals

Return consist the following -
(i) Copy back outmode parameter.
(ii) Deallocation the storage used for local variables.
(iii) Restore the execution status of the calling program unit.
(iv) Disable the access for non-locals.
(v) Return the control.

* Implementing simple subprograms

The semantic of FORTRAN 77 subprogram called require the following functions -
(i) Save the execution status of the current programm need.
(ii) Carring out the parameter passing process.
(iii) Pass the return address to the callee.
(iv) Transfer control to the callee.

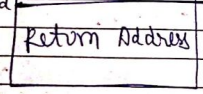
The Semantic of FORTRAN 77 subprogram return require the following functions -
(i) A pass by value result parameter are passed, the current value of those parameters are move to corresponding actual parameters.
(ii) If the subprogram is a function the function value is move to corresponding actual parameters.
(iii) The execution status of the caller is restored.

Spiral

Control is transfer back to the caller.

The call and return actions require the storage for the following:-
(i) Status information about the caller.
(ii) Parameter and return address.
(iii) Functional value for function subprogram.

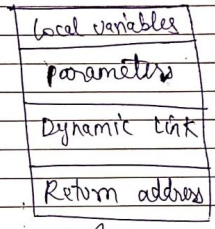
The format or layout of non-code part of an executing subprogram is called an activation record.
An activation record is a concrete example of a particular subprogram activation. The form of activation record is static and activation record is an instance for an FORTRAN 77 subprogram has a fixed size it can be statically allocated. an infact it could be attached to the code part of the subprogram.



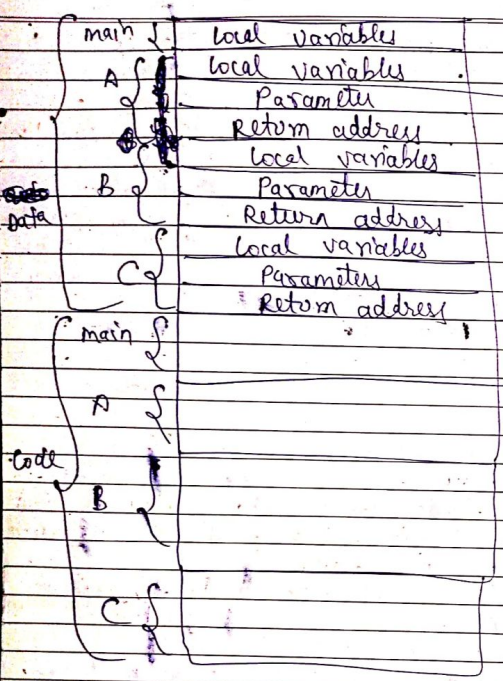
* Implementing subprograms with stack-dynamic local variables & activation record

The subprogram with stack dynamic local variables consist the parameters that can usually passed by two different methods as pass by value and pass by reference.

The compilers must generate the code to implicit allocation and deallocation of local variables and recursion must be supported as the possibility of multiple simultaneous activation of a subprogram.
An activation record keep the track of values as the program execution.



An activation record consist the fields as local variables, actual parameters, control and access link. and return value.
An activation record instance is dynamically created when a subprogram is called
Activation record instance, reside on the run time stack
The environment pointer (EP) or frame pointer (FP) must be maintained by the run time system. They always point at the base of activation record instance of the currently executing program units.



Code & Activation Record of FORTRAN 77 program

A FORTRAN 77 program consist of common storage, main program and three subprogram A, B and C. It showed all the

Spiral

Date

separated from

code segment all the activation record instance But in some cases these are attached to these associated code segment. The four program units main, A, B, C may have been compiled and executed in diff ways at the time each unit is compiled and machine code along with a list of reference to external subprogram and common variables within the code. and the executable program put together by the linker which is the part of O.S.

* Nested subprograms

But some non C based static scope languages as FORTRAN 95, Ada, JavaScript, Python, Ruby use the stack dynamic local variables and allow the subprogram to be nested. All variables that can be non-locally access reside in some activation record instance in the stack. The process of locating a non-local reference do the following -

1) Find the correct activation record instance
2) determine the correct offset within that activation record instance.

Locating a non-local reference as finding the offset is easy and find the correct activation record instance.
The static semantic rules, guaranteed all non-local variables that can be referenced

Spiral

that have been allocated in some activation record instance that is on the stack. when the reference is made.

* Implementing Dynamic scoping

There are atleast two ways in which non-local reference in a dynamic scope language can be implemented.

(i) Data access (Deep access)

When a program in a language that use dynamic scoping refers to a non-local variable the reference can be resolved by searching through the declaration in the other subprograms that are currently active. The dynamic chain linked together all subprogram activation record instance in the reverse of the order in which they are activated. The dynamic chain is exactly what is needed to referenced the non-local variables in a dynamic scope languages. This method is called Deep Access, bcz access may require search the deep in the stack.

(ii) Shallow Access

It is an alternative method where the variable declaration in subprogram are not stored in the activation record.

Spiral

For implementing the shallow access to use a central table that has a location for each different variable name in the program. Along with each entry a bit called active that indicate whether the name has a current binding or variable association.

Any access to any variable can be an offset into the central table. The offset is static so that access can be fast.

The maintenance of a central table in a subprogram call require that all of its local variables logically placed in the central table. If the position of the new variable in the central table is already active and if it contain a variable whose lifetime has not yet ended that value must be save somewhere during the lifetime of new variable.

When the lifetime of the local variable can be nested and work efficiently for the entire program during the execution.

Spiral

Unit-4

Date

Object oriented Programming (OOPs)

Object-oriented programming is a paradigm that provide many concepts as object to design the apps and computer programs. The programming paradigm where everything is represented as an object is known as object oriented programming language.

The main aim of OOPs is to bind the data and different types of functions that operate on them. So that no other part of the code can access this data except that function. Following are the features of OOPs.

(i) Object — An entity that have state and behaviours is known as an object. It can be physical or logical.

(ii) Class — It is a collection of object. It is a logical entity. It is a user defined data type which holds its data members and member functions.

```
class <class name>
{
  Data
  function
};
```

Spiral

Date

```
main()
{
  Class name object name1, object name2 ;}
```

For ex- There may be many class with different name but all of them will share some common properties like four wheels, speed limit, mileage, range etc.

3- Inheritance

When one object acquire all the properties and behaviours of parent object it is known as inheritance. The capability of a class to derive the properties from another class. It provide the code reusability. It is used to achieve the run time polymorphism.

4. Polymorphism

It means having many forms and has the ability of a message to be display in more than one form.

5- Abstraction

Hiding the internal detail and showing the functionality is known as abstraction. In C++, we use abstract class and interface to achieve an abstraction.

Spiral

Date

It to providing the essential info. about the data to the outside world and hiding the background detail for implementation.

6. Encapsulation

The wrapping or binding the code and data together into a single unit is known as encapsulation.

* Access Modifiers in C++

Access modifiers is used to identify access right for the data and member functions. of the class. There are three main types of access modifiers or access specifier in C++ programming language.

1. Private - The private member is inaccessible from outside the class means only member of same class are allowed to access.

2. Public - Everyone is allowed to access.

3. Protected - It is a stage b/w private and public access. If member functions defined in a class are protected they cannot be accessed from outside the class but can be accessed from the derived class.

Spiral

Date

* C++ Data types

1. Primitive Data types
   - Char
   - bool
   - int
   - Float

2. Derived Data types
   - Array
   - function
   - Pointer

3. User or Abstract Datatypes
   - class
   - Structure
   - union
   - Enumeration, Typedef

* Function overloading

It is defined as the process of having two or more function with the same name but different in parameters. The function is redefined by using other different types of arguments or different no. of arguments.

The advantage of this is that increase the readability of the program bcz you don't need to use different names for the same action.

Spiral

Date

```
#include <iostream>
using namespace std;
class cal
{
public:
static int add(int a, int b)
{
  return a+b;
}
static int add(int a, int b, int c)
{
  return a+b+c;
}};
int main(void)
{
  cal c;
  cout<<c.add(10,20)<<endl;
  cout<<c.add(10,20,25);
  return 0;
}
```

output 30
55

* operator overloading in c++

In C++, we can operator to work for user-defined classes. C++ has the ability to provide the operator with a special meaning for a data type - This ability is known as operator overloading.

Spiral

Date

An operator is overloaded to give user defined meaning to it.

Allocation Deallocation
C++ new() delete()
C malloc() free()

```
#include<iostream>
using namespace std;
class Test
{
private:
  int count;
public:
  Test();
  count(5);
};
void operator++(C)
{
  count = count+1;
}
void display()
{
  cout<<"count:"<<count;
};
int main()
{
  Test t;
  ++t;
  t.display();
  return 0;
}
```

Spiral

Date.....

* Friend function

The friend function of a class is defined outside that class scope but it has the right to access all private and protected members of the class. Even though the prototype for friend function appear in the class definition.

The advantage of friend function do not violate encapsulation bcz the class itself decide which function are its friend.

* Function overriding

It is a feature that allow us to have a same function in child class which is already present in the parent class. A child class inherit the data members and member function of parent class but you when you want to overwrite a functionality in the child class then you can use function overriding.

* Constructors in c++

A constructor is a member function of a class which initialise the object of a class. It has the same name as its class. Constructor is automatically called when an object create it is a special function of the class.

Spiral

Date.....

It is of 3 types -

(i) Default constructor

It has no arguments in it. For ex

```
#include<iostream>
using namespace std;
class Test
{
public:
  int a,b;
  Test()
  { a=10;
    b=20;
  }};
int main()
{
  Test c;
  cout<<"a:"<<c.a<<endl;
  cout<<"b:"<<c.b<<endl;
}
```

output
a=10
b=20

// write a program for simple calculator in c++

```
#include<iostream>
using namespace std;
void main()
{ int a,b,c,d;
  cout<<"Enter any no";
  cin>>a>>b;
  cout<<"Enter the operator for operation('+','-','*');
```

Spiral

```
cin >> d;
if(d == '+')
{ c = a+b;
  cout<< c;
}
else if( d == '-')
{ c = a-b;
  cout<< c;
}
else if( d == '*')
{
  c = a*b;
  cout<< c;
}
else if(d == '/')
{
  c= a/b;
  cout<< c;
}
else if(d == '%')
{
  c= a%b;
  cout<< c;
}
else
{ cout<< "You have entered wrong";}
}
```

Spiral

Spiral