**1.What are the control sequences in Ada?**

Control sequances in ada include If statements, while and for loops, case statements and assignments. Example:

if condition then while cond loop

do\_something; so\_smth;

end if; end loop;

**2.What are the programming units in Ada?**

Programming units in ada are packages, functions, and procedures. Packages are used to encapsulate data

**3.What are the differences between a procedure and a function?**

There are differences in syntax and in semantics. In syntax, they have different type signature. A function should have a return value, while a procedure should not. We can declare a function like this: function F(parameter: in typeIn) return typeOut; and a procedure like this : procedure P(param: in out Type);

Semantic, the return value has to be given to a variable, so the function should return something, while procedures communicate only via parameters. On the original semantics, a function could have only in parameters while a procedure can have all the types so: in, out and in out. In

**4.What type of parameters can be defined in Ada? Can we define default parameter values? How?**

There are three different types of parameters: in ,out and in out.

In: The actual parameter value goes into the call and is not changed there. The formal parameter is a constant and allows only reading. This is the default when no mode is given. The actual parameter is an expression.

Out: The actual parameter's value before the call is irrelevant, it will get a value in the call. The formal parameter can be read and written.

In out : The actual parameter goes into the call and may be redefined. The formal parameter is a variable and can be read and written.

We can define default parameters. For example in : function add(x: in Float:=3.4); after we define it we can call it as add() and the function will take x as 3.4 .

Types can also have parameters and they are called discriminants and they can also have default value. Example :

type my\_record(Max: Positive:=10 ) is record

Data: TArray(1..Max);

Pointer: Natural := 0;

end record;

**5. How can we declare pointers?**

First we have to declare the type of which the pointer will point to. And then declare any pointer of that type. Example :

type Person is record

First\_Name : String (1..30);

Last\_Name : String (1..20);

end record;

type Person\_Access is access Person;

and the on the demo we could use it as P:Person\_Access;

Dereference can be done by .all , so for example:

P.all.First\_Name:=”Tringa”;

**6. What is a block structure? When can be used?**

The block structure is of type

Declare

Begin

End;

3 typical advantage : delayed declaration of a variable for example you have to read the size of a matrix and then declare the matrix or you have a large data set that you want to use for a short time then you declare it in a block structure because all the values will be deleted after the end of that block structure. And lastly to modify the visibility so that exception handling can be done.

Example:

Declare…

Begin ..

Ada.Integer\_Text\_IO.Get(n);

Declare

Mat: arr(1..n,1..n);

Begin

…

End;

End;

**7. Why is called Ada as strongly typed language?**

Being a strongly typed language mean that types are incompatible with each other even if they belong to the same family. In ada, we can create new types. And sometimes, even though the new type is just a renaming of the old one, we cannot mix them(they are non-equivalent) with each other, or even mix different new types derived from the same type.

Subtyping on the other hand does not create an entire new class but its usually used to introduce some sort of constrain. For example if we want to implement a rational type, for the denominator, instead of using a natural we use a positive to assure that it wont be zero. While subtypes of a type are statically compatible with each other, constraints are enforced at run time: if you violate a subtype constraint, an exception will be raised.

Subtyping,new typing issue about speed distance non equivalence between new type and old type.

**8.What is a package? What is the structure of a package?**

Packages are program units that allow the specification of groups of logically related entities. A package is like an interface. We put the specification in an .ads file and the body/implementation in an .adb file. It helps encapsulation the data together with the operations and hide the implementation depending if you want some variables or operations to be public or private. You can also declare use a limited type in the package, which limit the usage of assignment and equality checks;

A package generally consists of two parts, the specification and the body. A package specification can be further divided in two logical parts, the visible part and the private part. Only the visible part of the specification is mandatory. The private part of the specification is optional, and a package specification might not have a package body—the package body only exists to complete any incomplete items in the specification. Subprogram declarations are the most common incomplete items. There must not be a package body if there is no incomplete declaration, and there has to be a package body if there is some incomplete declaration in the specification.

**9.What are the differences between a subtype and a new type?**

New type introduces a new type which is not acquainted with the base type, meaning they create a set of values which are distinct from the parent. They require new I/O packages and cannot be assigned to another type without conversion.

Subtype is usually a subset of the type but don’t introduce a new type.

Subtype is generated subset of the base type, still being equivalent to base type. They can be used in place of parent types and are defined on the same set of data.They don’t require new I/O packages and can be assigned to a variable of the parent type or different subtypes of the same parent without conversion but constrained are checked at runtime.

Example of subtype : subtype MonthDay is Integer range 1 .. 31;

Example of new type : type MonthDay is range 1 .. 31;

**10.How can we define abstract data structures**?

By packages we give interfaces for typical operation that are related to adt. Give type related currying function. Give specification and body in a typical package.

Private,public,limited.

**11.Why and where are generics used?**

Generics is used to provide a template or scheme that has to be later instantiated. Usually when you don’t want to provide some implementation for each type separately so you provide one generic one which can be used with different types but has the same operation, like a general solution for a problem. Any programming unit can be generic, meaning we can have a generic package, function or procedure. The data type required is passed in as a parameter when the generic unit is instantiated.

**12.What type of parameters can be used at generics? How can we instantiate generics?**

The generic unit declares generic formal parameters, which can be:

-objects (of mode in or in out but never out)

-types (simple,composed)

-subprograms (function,procedure)

-instances of another, designated, generic unit.

When instantiating the generic, the programmer passes one actual parameter for each formal one in order. Formal values and subprograms can have defaults, so passing an actual for them is optional.

Example:

generic

type A is private; --elementary type

type B is private; --elementary type

type Index is (<>); --discrete type

type TA\_Array is array ( Index range <> ) of A; --composed type

type TB\_Array is array ( Index range <> ) of B; --composed type

with function Op(x: A) return B; --subprogram

function Map(ta: TA\_Array) return TB\_Array; --> generic function

we instantiate it like :

type t1 is array (Integer range <>) of Integer;

type t2 is array (Integer range <>) of Float;

function square (x: Integer) return Float;

function my\_map is new map(Integer, Float, Integer, t1, t2, square);

We can also have generic in generic like:

generic

type Elem is private;

package Stacksgen is

….

generic

with procedure Process\_Element(Item: in Elem );

procedure For\_Each( S: in Stack );

and when we want to instantiate it, first we instantiate the generic unit and then give it as a param to the main one.

**13.How can we defined default operators?**

If we take a look at this example :

generic

type T is private;

with function "<" (A, B: T) return Boolean is <>;

function Maximum ( A, B: T ) return T;

then the sign <> indicates that in case the operator is not given that use the default operator of the given type.

**14.Compare the type V is private; line in the two code snippets:**

• generic type V is private;

.... package VV is .....

end VV;

• package W is.....

type V is private;

...... (!)

end W;

In the first one type v is a generic type parameter, while in the second one v is a private type in W so implementation is hidden in the private part.

**15.How can we define records? What kind of records can be used?**

Composite types consisting of several fields of different types. Fixed records ->fixed structure records ->classical

Records with discriminant

**16. Write examples of arrays and enumerate their attributes.**

**Indefine ranges, index type,elem type attribute first last range**

**Matrix**

**17.What is a limited private type?**

Limited means equality and assignment cant be defined for that type

Private means the representation is hidden.

**18.Describe the exception handling in Ada.**

4 issues to touch

-how to define your own exception

Exceptions are declared similarly to objects. Empty\_Stack, Full\_Stack : Exception;

-how to handle error

When an exception occurs, the normal flow of execution is abandoned and the exception is handed up the call sequence until a matching handler is found. Any declarative region (except a package specification) can have a handler. The handler names the exceptions it will handle. By moving up the call sequence, exceptions can become anonymous; in this case, they can only be handled with the others handler.

-where to put them

- what are the classical error

Constraint\_Error, raised when a subtype’s constraint is not satisfied

Program\_Error, when a protected operation is called inside a protected object, e.g.

Storage\_Error, raised by running out of storage

Tasking\_Error, when a task cannot be activated because the operating system has not enough resources

-how is the error propagating and how is the exception found in the dynamic chain.

**19.How can we define parallel units in Ada?**

Define by task and protected. Write task type and singleton what are they used for , make parallel programming and to make a diff synchronization exchange info . entry singnature waiting queues

A task unit is a program unit that is obeyed concurrently with the rest of an Ada program. A task unit has both a declaration and a body, which is mandatory. A task body may be compiled separately as a subunit, but a task may not be a library unit, nor may it be generic. Every task depends on a master, which is the immediately surrounding declarative region - a block, a subprogram, another task, or a package. The execution of a master does not complete until all its dependent tasks have terminated.

Tasks may be singe :

task single;

task body single is

end single;

It is possible to declare task types, thus allowing task units to be created dynamically, and incorporated in data structures:

Task type t is

End t;

Task body t is

End t;

Task types are limited, i.e. they are restricted in the same way as limited private types, so assignment and comparison are not allowed.

We can have entries in tasks for synchronisation. Entries are declared in the task specification.

Each task entry can have one or more accept statements within the task body. If the control flow of the task reaches an accept statement, the task is blocked until the corresponding entry is called by another task.

**20.What is a protected unit in Ada?**

It’s a programming unit that is good for protecting distributed variable. A distributed variable is a shared variable that multiple tasks can use. Automatic mutual exclusion is assured for these shared variables. In it we can have a function, procedure, and an entry. An entry is a condition for making rendez-vous and it always has a queue. A function only delivers information about a shared variable while a procedure and an entry can modify it. They are very useful for writing semaphores, monitors and protected printings, so synchronization mechanisms.

**21. How can we define a rendez-vous in Ada?**

Meeting beewtween two tasks protected. How is it done.

Entry points in normal tasks and protected. It’s asynchronous, each of them needs to wait for the partner for the rendex-vous, its asynchronous in the way in a way that it has to be a one to one meeting you cannot skip it, its infinitely supspended if nobody calls.

-----on the internet:

The only entities that a task may export are entries. An entry looks much like a procedure. It has an identifier and may have in, out or in out parameters. Ada supports communication from task to task by means of the entry call. Information passes between tasks through the actual parameters of the entry call. We can encapsulate data structures within tasks and operate on them by means of entry calls, in a way analogous to the use of packages for encapsulating variables. The main difference is that an entry is executed by the called task, not the calling task, which is suspended until the call completes. If the called task is not ready to service a call on an entry, the calling task waits in a (FIFO) queue associated with the entry. This interaction between calling task and called task is known as a rendezvous. The calling task requests rendezvous with a specific named task by calling one of its entries. A task accepts rendezvous with any caller of a specific entry by executing an accept statement for the entry. If no caller is waiting, it is held up. Thus entry call and accept statement behave symmetrically.

**22.When are the tasks activated? When are the tasks terminated?**

Activation : if static task then its activated right after its declaration finishes and if its dynamic then at the point we created the pointer.

Ternimated : they termnate by finishing execution of statements, with terminate included in select, which are considered normally. Or it can finish with abort meaning someone send a kill signal. You can abort yourself. Via exception handler – if no partner cuz it terminated. With fault or error meaning its never terminating because it has a deadlock, becayuse wait infinitely for a resource known as starving deadlock and it cannot finish an activity (livelock) or wait infinitely for a partner.

**23. Describe static and dynamic tasks.**

For static tasks, activation starts immediately after the complete elaboration of the declarative part in which they are defined.

Dynamic tasks are activated immediately after the evaluation of the allocator (the new operator) which created them.

Static can be single task object that we define in the declaration. Start at the end of declaration part. Task type and create object of that type you can write tasks with discrimination

**24. Describe the different select command types.**

1. Receiver : 1.braches with nothing, branches or delay, branches or terminate, branches delay until and branches else (immediate)

2. Caller : timed (or delay),immediate (else)

**25.What is a semaphore?**

It’s a synchronization mechanism to assure mutual exclusion for the critical section.

Two entry points declared : Entry and exit . can be single semaphore only one task is protected of its critical section by the semaphore . Can be multiple, has a variable which states number of tasks allowed in critical section.

Every answer should include small examples!!!