UNIT- III

TYPE CHECKING :

Type checker is a module of a compiler which is used to help the type checking tasks.

Type checking is the process of verifying that types of all variables, expressions and functions are according to the rules of the programming language.

TASKS :

1) Has to allow "indexing is only on an array".

2) Has to check the range of datatypes used.

3) INTEGER (int) has range of
-32,768 to +32,767

4) FLOAT has the range of
1.2E-38 to 3.4E+38

TYPES OF CHECKER :

1. static
2. Dynamic

1. STATIC TYPE CHECKING : - Pascal / C

It is defined as type checking performed at compile time. It checks the type variables at compile - time, which means type of variables are known as the compile - time. → determine the amount of memory needed to store the variab

Example : Name related - checks : Sometimes the same name may appear two or more times. For example, A loop may have a name that appears at the beggining and end of the construct. The compiler must check that same name is used at both places.

2. DYNAMIC TYPE CHECKING :

It is defined as type checking performed at run - time. In Dynamic checking, types are associated with values, not variables.

PURPOSE :

1) It is used to check the correctness of the program before its execution.

2) The main purpose of type checking is the check the correctness & data type assignments and type - casting of the data types.

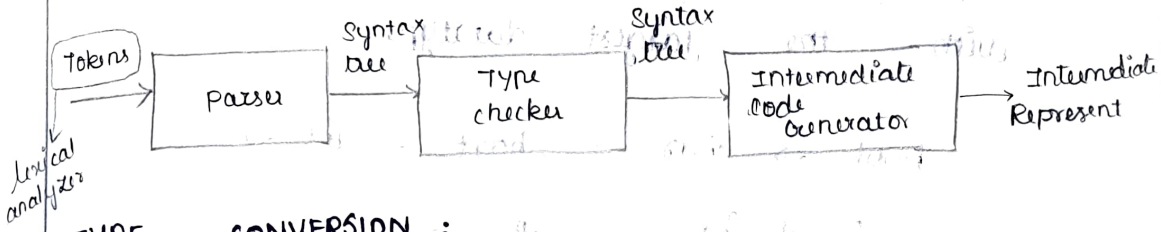
DESIGN OF TYPE CHECKER DEPENDS ON ;

1) syntatic, structure of language construct

2) The Expression of languages.

3) The rules for assigning type of constructs. (semantic rules)

POSITION OF TYPE CHECKER :



TYPE CONVERSION :

It is a process of converting one data type to another. The type of conversion is only performed to those data types where conversion is possible.

It is performed by compiler. The destination datatype can't be smaller than the source data type. It is also called widening conversion.

2 TYPES OF CONVERSION :

1. Implicit type conversion
2. Explicit type conversion

IMPLICIT TYPE CONVERSION :

It is also known as "automatic type conversion."

It has some rules ;

A. Done by compiler on it own, without any external trigger from the user.

B. Generally takes places when in an expression more than one datatype is present In such condition type conversion takes place to avoid loss of data.

C. All the datatype of variables are upgraded to the datatype of variable with the largest datatype.

bool → char → short → int →

unsigned int → long →

D. It is possible for implicit conversion to lose information, sign can be lost (signed → unsigned) and overflow can occur (long → float)

EXPLICIT DATA TYPE :

It is also called as type casting and it is user defined.

lower datatype —Explicit→ Higher data type

EQUIVALENCE OF TYPE EXPRESSION :

The type of a language construct is denoted by a type expression.

A type expression can be ;

i) A basic type.

Primitive datatype

type - error

void

ii) A type name,

A name can be used to denote a type expression.

iii) A type constructor applies to other type expressions.

→ Arrays : If T is type expression, then array (I, T) is a type expression where I denotes index ranges. Eg: Array (0.99, float).

→ Product : If T_1 & T_2 are type expression, their cartesian product $T_1 \times T_2$ are type expression

Eg: int x int

→ Pointers : pointer (T) Eg: pointer (int)

→ Function : Eg: int → int.

TYPES OF EQUIVALENCE TYPE EXPRESSION :

1. Structural Equivalence
2. Name Equivalence.

STRUCTURAL EQUIVALENCE :

If the type expressions are built from basic types and constructors then those expressions are called structurally equivalent.

Eg.

S_1	S_2	Equivalence	Reason
char	char	$S_1 = S_2$	Same basic type
Pointer (char)	pointer (char)	$S_1 = S_2$	similar (same) constructor ptr to the char equivalent type

NAME EQUIVALENCE :

Two type expressions are 'name equivalent' if and only if they are identical, they can be represented by same syntax tree with same labels.

FUNCTION OVERLOADING :

→ Multiple function can have a same name but different parameters.

→ This allows one function to perform different tasks depending on the context of call

→ The function must differ either by number or type of parameters.

HOW FUNCTION OVERLOADING WORKS ?

NUMBER OF PARAMETERS : → differing no of parameters

Eg :
```
void display ( int i )
void display ( int i , int j )
```

TYPE OF PARAMETERS : → same no of parameters as long as types are different

Eg :
```
void display ( int i)
void display ( double d )
```

ORDER OF PARAMETERS : → no's & type of parameter are same their order can be changed

Eg :
```
void display ( int i , double d )
void display ( double d , int i)
```

OPERATION OVERLOADING :

→ Different implementation to a single operator based on the datatypes on which they are operated.