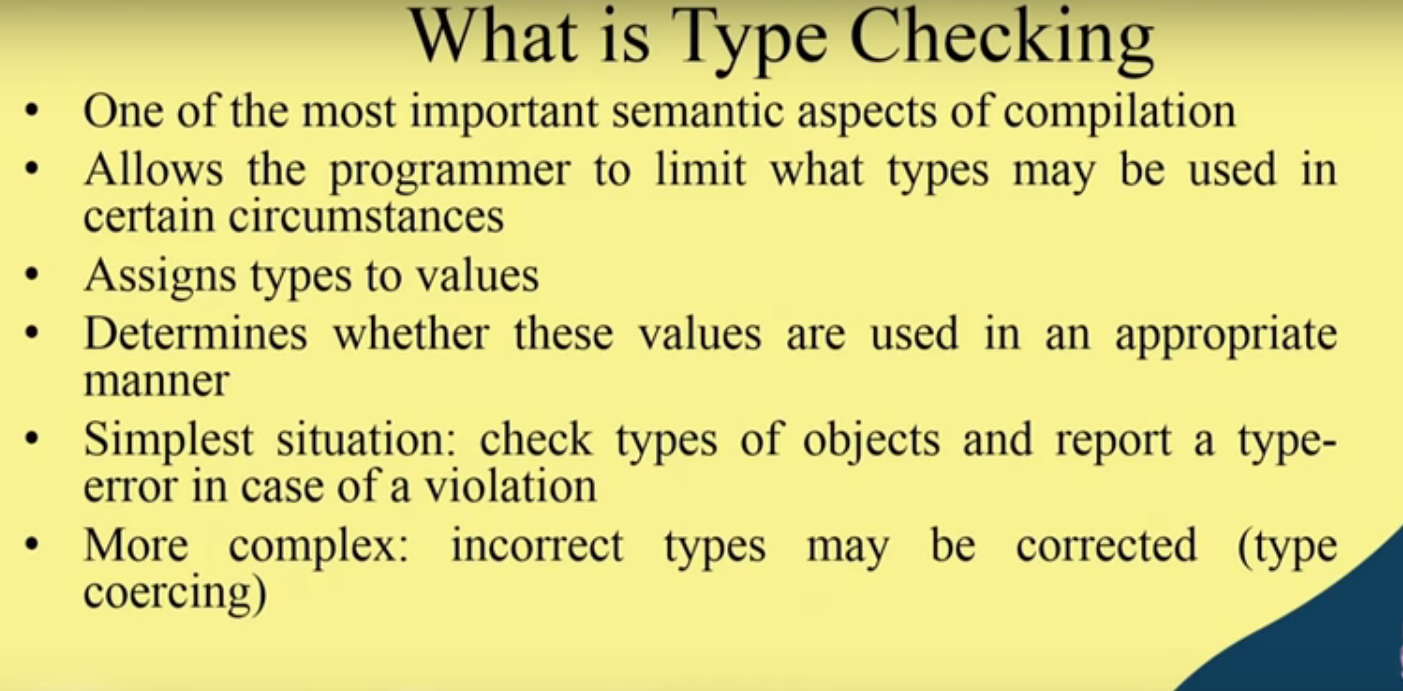
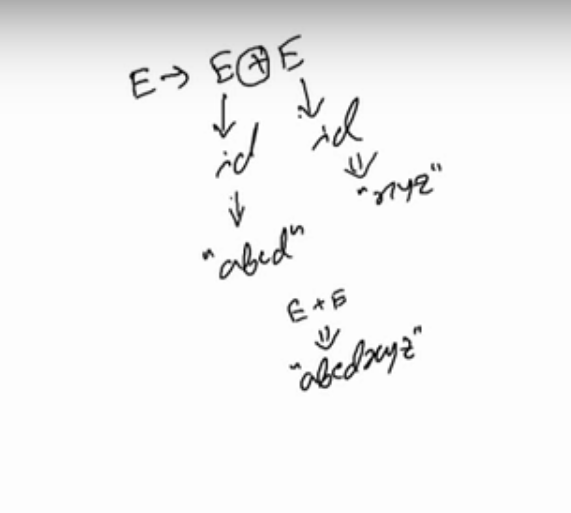
**Type Checking**

1. **What is type Checking**



Example 1)



Example 2)

A=B\*j+d;

\* and - are basically int and float data types based operations and if any variable in this A=B\*j+d;Is of other than int and float then compiler will generate type error.

1. **Static vs Dynamic Checking ( Types of type checking)**
2. **Static Type**

* Static Type Checking is done at complete time.
* Information needed at compile time is provided- by declaration- by language structures.
* The information required includes:

1) for each operation: The number, order, and data type, of its arguments.

2) For each variables: Name and data type of data object.

Example-

A+B

**in this type of A and B variables must not be changed.**

3) for each constant: Name and data type and value

const int x=28;

const float x=2.087;

**In this data type, the value and name is specified and in further if checked value assigned should match its data type.**

**Advantages of Static Type Checking:**

1) compiler saves information:- if that type of data is according to the operation then compiler saves that information for checking later operations which further no need of compilation.

2) checked execution paths: As static type checking includes all operations that appear in any program statement, all possible execution paths are checked, and further testing for type error is not needed. So no type tag on data objects at run-time are not required, and no dynamic checking is needed.

**Disadvantages of Static Type Checking**

: It affects many aspects of languages

1) declarations

2) data control structures

3) provision of compiling separately some subprograms.

**b) Dynamic Type Checking:**

• It is done at runtime.

• It uses concept of type tag which is stored in each data objects that indicates the data type of the object.

Example:

**An integer data object contains its 'type' and 'values' attribute.**

**So Operation only be performed after type checking sequence in which type tag of each argument is checked. If the types are not correct then error will be generated.**

• Perl and Prolog follow basically dynamically type checking because data type of variables A+B in this case may be changed during program execution.

• So that type checking must be done at runtime.

**Advantages of Dynamic Type:**

• It is much flexible in designing programs or we ca say that the flexibility in program design.

• In this no declarations are required.

• In this type may be changed during execution.

• In this programmerare free from most concern about data type.

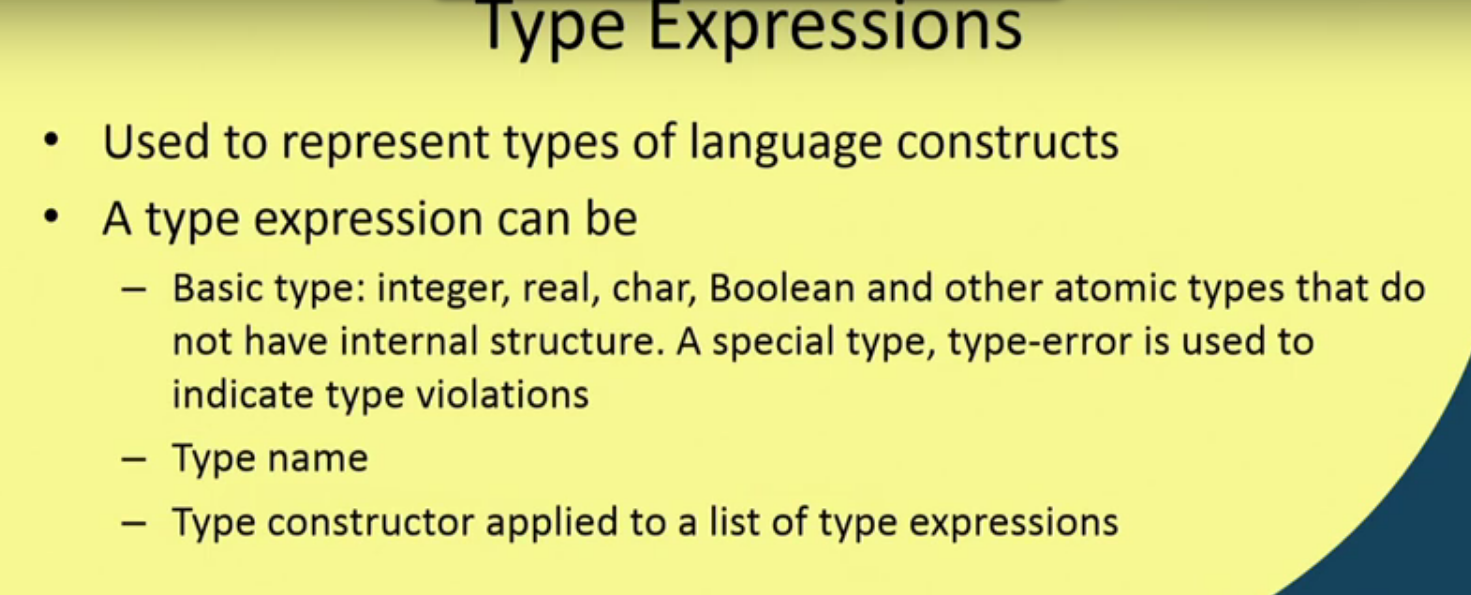
**Disadvantage of Dynamic Type:**

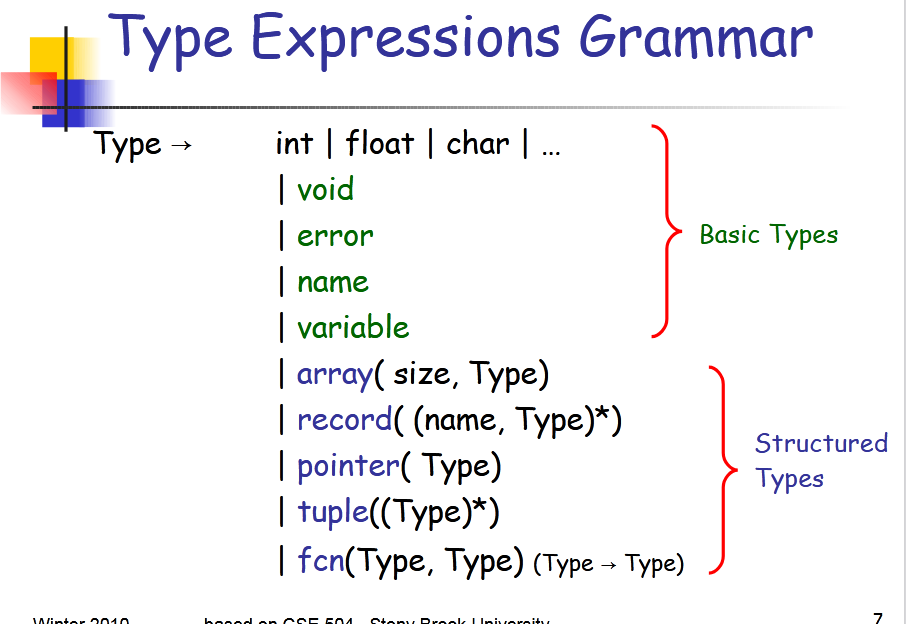
• Difficult to debug: We need to check program execution paths for testing and in dynamic type checking, program execution path for an operation is never checked.

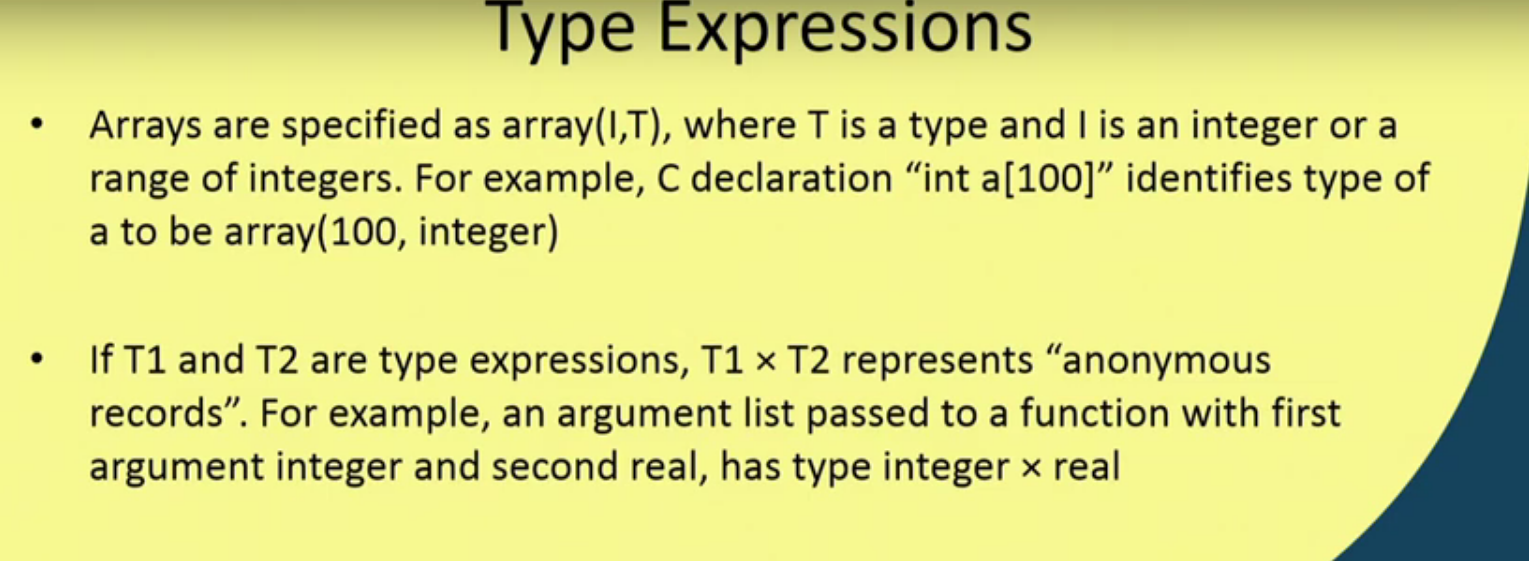
• Extra storage: Dynamic type checking need extra storage to keep type information during execution.

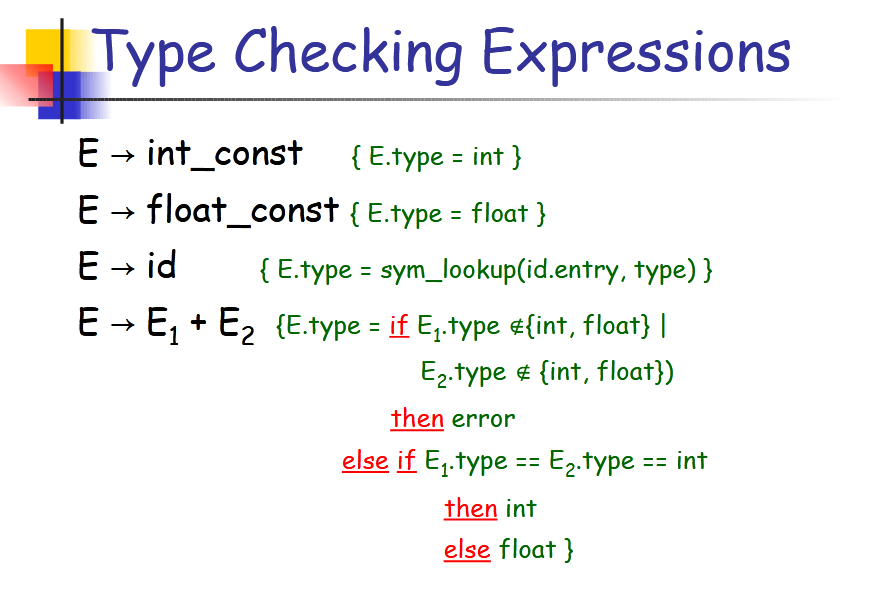
• Seldom hardware support : As hardware seldom support the dynamic type checking so we have to implement in software which reduces execution speed.

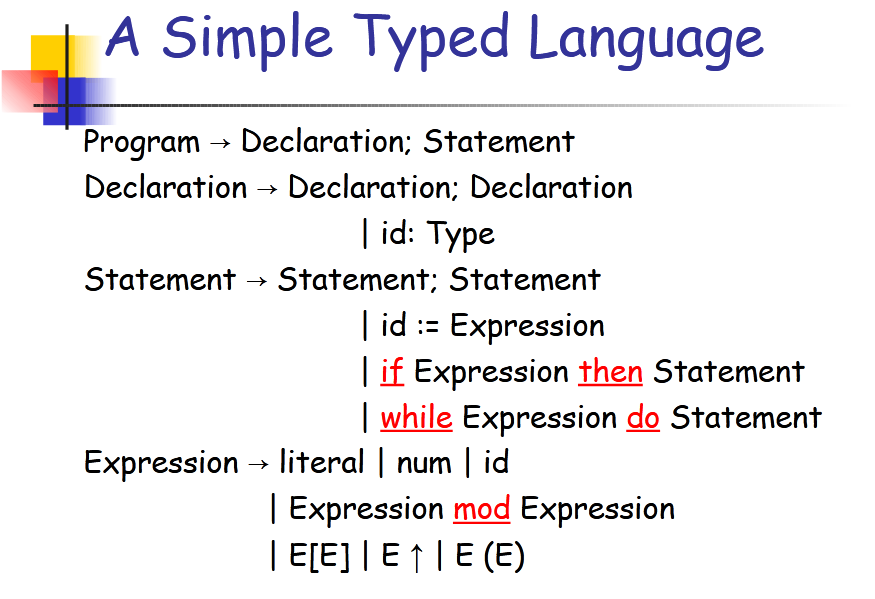
1. **Type Expression**











1. **Type Equivalence**

TYPE CHECKING RULES usually have the form

**if** two type expressions are equivalent

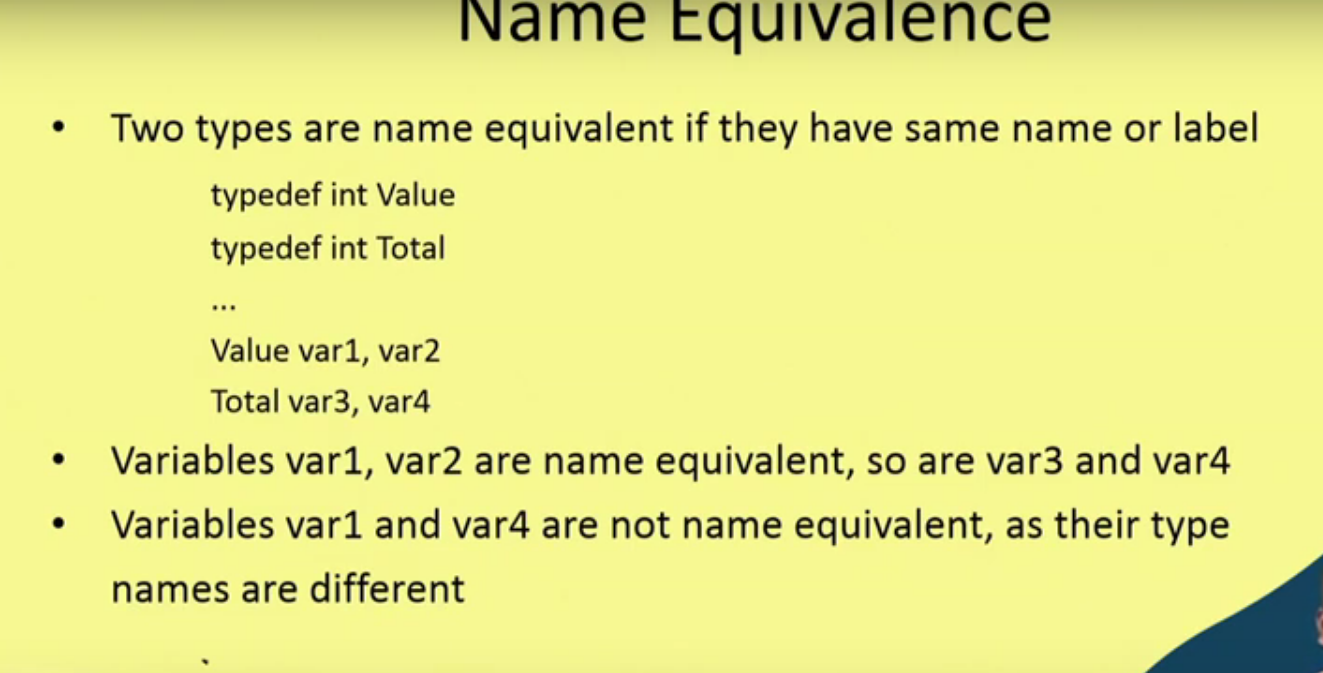
**then** return a given type

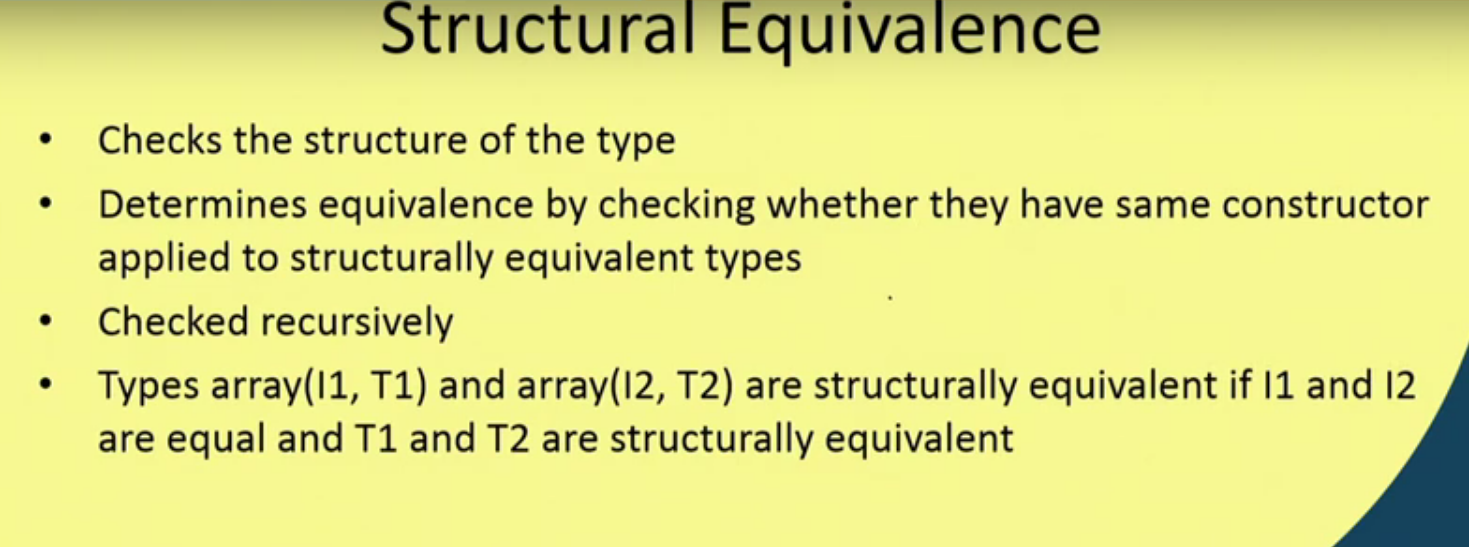
**else** return **type\_error**

**OR**

**Equivalence of Type Expressions**

* If two type expressions are equal then **return** a certain type else return type\_error.
* **Key Ideas:**
  + The main difficulty arises from the fact that most modern languages allow the naming of user-defined types.
  + For instance, in C and C++ this is achieved by the typedef statement.
  + When checking equivalence of named types, we have two possibilities.
    - Structural Equivalence
    - Names Equivalence



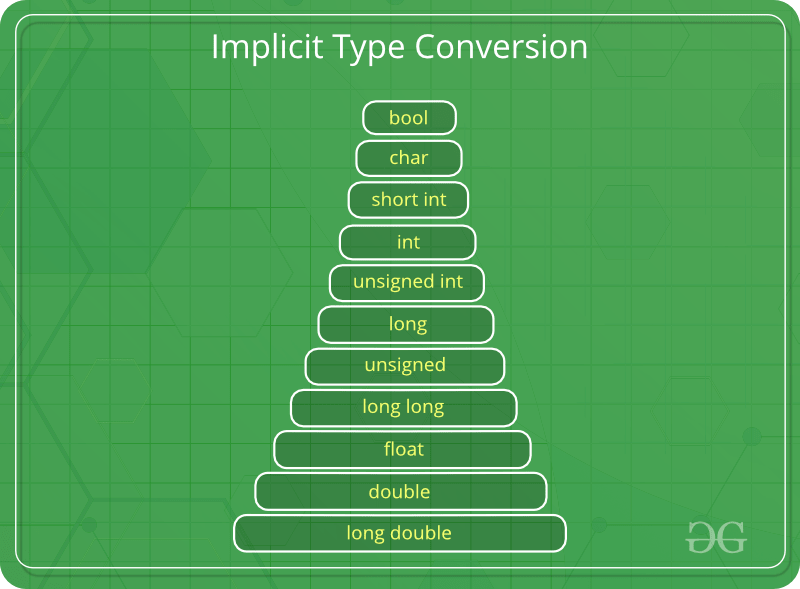


1. **Type Conversion**

Type Conversion in C

A type cast is basically a conversion from one type to another. There are two types of type conversion:

1. **Implicit Type Conversion**



Also known as ‘automatic type conversion’.

* + Done by the compiler on its own, without any external trigger from the user.
  + Generally takes place when in an expression more than one data type is present. In such condition type conversion (type promotion) takes place to avoid loss of data.
  + All the data types of the variables are upgraded to the data type of the variable with largest data type.
  + **bool -> char -> short int -> int ->**
  + **unsigned int -> long -> unsigned ->**
  + **long long -> float -> double -> long double**
  + It is possible for implicit conversions to lose information, signs can be lost (when signed is implicitly converted to unsigned), and overflow can occur (when long long is implicitly converted to float).

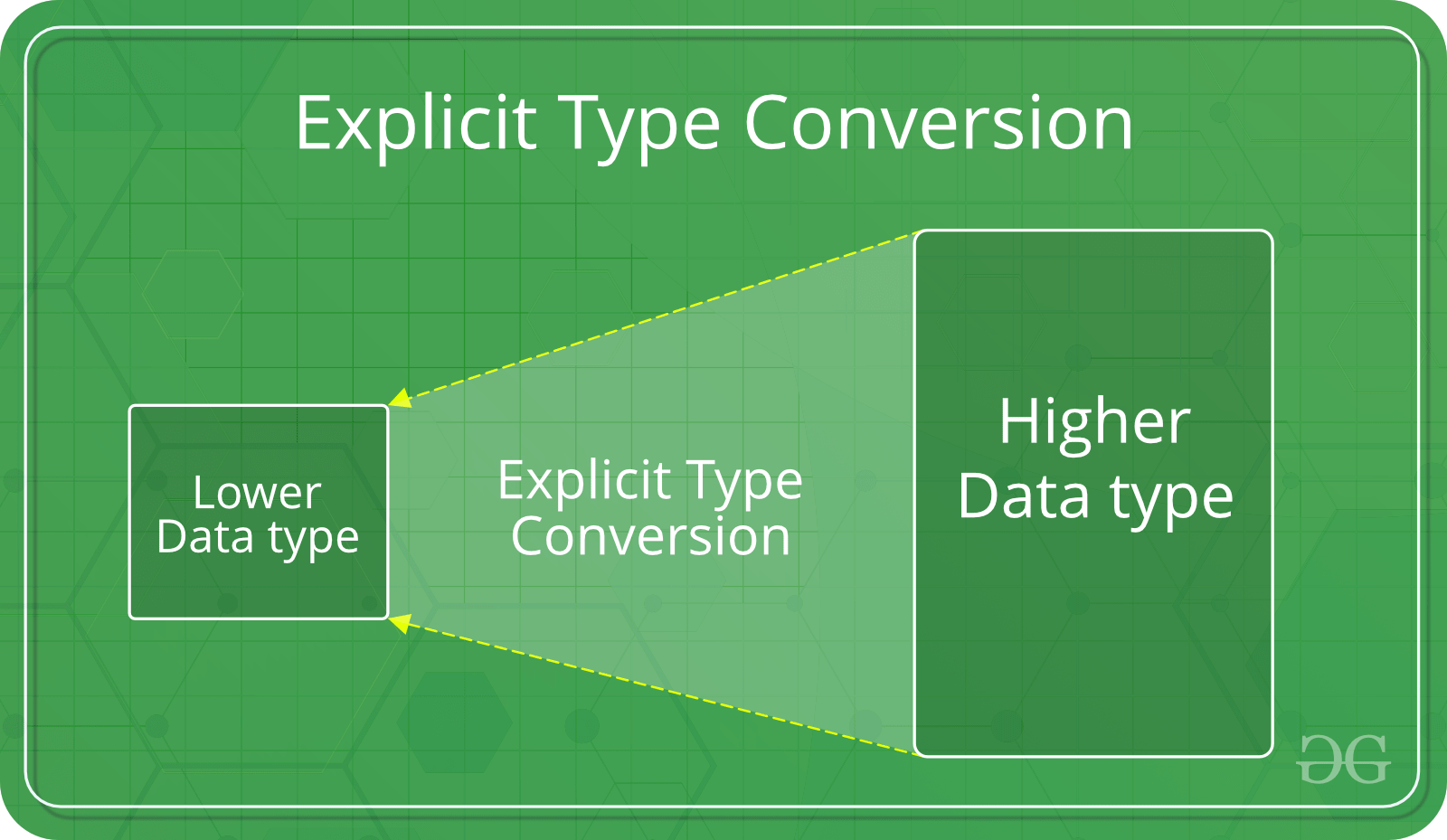
**Example of Type Implicit Conversion:**

|  |
| --- |
| // An example of implicit conversion  #include<stdio.h>  int main()  {      int x = 10;    // integer x      char y = 'a';  // character c        // y implicitly converted to int. ASCII      // value of 'a' is 97      x = x + y;        // x is implicitly converted to float      float z = x + 1.0;        printf("x = %d, z = %f", x, z);      return 0;  } |

Output:

x = 107, z = 108.000000

1. **Explicit Type Conversion**–



This process is also called type casting and it is user defined. Here the user can type cast the result to make it of a particular data type.

The syntax in C:

(type) expression

Type indicated the data type to which the final result is converted.

|  |
| --- |
| // C program to demonstrate explicit type casting  #include<stdio.h>    int main()  {      double x = 1.2;        // Explicit conversion from double to int      int sum = (int)x + 1;        printf("sum = %d", sum);        return 0;  } |

Output:

sum = 2

Advantages of Type Conversion

* + This is done to take advantage of certain features of type hierarchies or type representations.
  + It helps us to compute expressions containing variables of different data types.

https://www.youtube.com/watch?v=p-hJBrfmPGU