TYPE INFERENCE IN C++

- Type Inference refers to **automatic deduction of the data type of an expression** in a programming language.
- All as all the types are deduced in compiler phase only, the time for compilation increases slightly but it does not affect the run time of the program.
- As all the types are deduced in compiler phase only, the time for compilation increases slightly but it does not affect the run time of the program.
- **AUTO**
  - ✓ The auto keyword specifies that the type of the variable that is being declared will be automatically deducted from its initializer. In case of functions, if their return type is auto then that will be evaluated by return type expression at runtime.
  - ✓ **The variable declared with auto keyword should be initialized at the time of its declaration only or else there will be a compile-time error**

```
// C++ program to demonstrate working of auto
// and type inference
#include <bits/stdc++.h>
using namespace std;

int main ()
{
        // auto a; this line will give error
        // because 'a' is not initialized at
        // the time of declaration
        // a=33;

        // see here x, y,ptr are
        // initialised at the time of
        // declaration hence there is
        // no error in them
        auto x = 4;
        auto y = 3.37;
        auto ptr = &x;
        cout << typeid(x).name () << endl
                << typeid(y).name () << endl
                << typeid(ptr).name () << endl;

        return 0;
}
```

Output

```
i

d

Pi
```

- A good use of auto is to avoid long initializations when creating iterators for containers.

```
// C++ program to demonstrate that we can use auto to
// save time when creating iterators
#include <bits/stdc++.h>
using namespace std;

int main()
{
    // Create a set of strings
    set<string> st;
    st.insert({ "geeks", "for",
                "geeks", "org" });

    // 'it' evaluates to iterator to set of string
    // type automatically
    for (auto it = st.begin();
          it!= st.end(); it++)
          cout << *it << " ";

    return 0;
}
```

Output

```
for geeks org
```

Note: auto becomes int type if even an integer reference is assigned to it. To make it reference type, we use auto &.

```
// function that returns a 'reference to int' type

int& fun () {   }


// m will default to int type instead of

// int& type

auto m = fun ();


// n will be of int& type because of use of

// extra & with auto keyword

auto& n = fun ();
```

- **decltype Keyword**

  It inspects the declared type of an entity or the type of an expression. Auto lets you declare a variable with particular type whereas decltype lets you extract the type from the variable so decltype is sort of an operator that evaluates the type of passed expression.

```
// Another C++ program to demonstrate use of decltype
#include <bits/stdc++.h>
using namespace std;
int main ()
{
    int x = 5;

    // j will be of type int: data type of x
```

```
            decltype(x) j = x + 5;

            cout << typeid(j).name ();

            return 0;
      }
```

Output

```
i
```

decltype vs typeid:

decltye gives the type information at compile time while typeid gives at runtime. So, if we have base class reference (or pointer) referring to (or pointing to) a derived class object, the decltype would give type as base class reference (or pointer, but typeid would give the derived type reference (or pointer).