Auto Keyword in C++

The auto keyword specifies that the type of the variable that is being declared will be automatically deducted from its initializer. In the case of functions, if their return type is auto then that will be evaluated by return type expression at runtime.

Note: 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.

```
#include <iostream>
#include <typeinfo>
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
    // initialized 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;
    cout << typeid(y).name() << endl;
    cout << typeid(ptr).name() << endl;

    return 0;
}
```

Output

```
i
```

```
d
Pi
```

> 👉 Note: We have used typeid for getting the type of the variables.

Typeid is an operator which is used where the dynamic type of an object needs to be known.

typeid(x).name() returns the data type of x, for example, it return 'i' for integers, 'd' for doubles, 'Pi' for the pointer to integer etc. But the actual name returned is mostly compiler dependent.

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();*

Advantages -

1)No Conversion happens when auto is used

```
float x = 3.4 //double literal converted to float.    auto
y = 3.4 //type of y is double.
```

2)If a function returns auto, its return type can be changed without worrying about prototype change.

3)Can be very helpful for lengthy types, especially STL iterators

```
vector<int>::iterator i;can be replaced withauto i;
```

4)It can be used in Lambda Expressions

Disadvantages

The main drawback is that, by using auto, you don't necessarily know the type of object being created. There are also occasions where the programmer might expect the compiler to deduce one type, but the compiler adamantly deduces another. It takes more processing time.