Type Inference in C++ (auto and decltype)

Difficulty Level : Easy • Last Updated : 02 Mar, 2022

Type Inference refers to automatic deduction of the data type of an expression in a programming language. Before C++ 11, each data type needed to be explicitly declared at compile-time, limiting the values of an expression at runtime but after the new version of C++, many keywords are included which allows a programmer to leave the type deduction to the compiler itself.
With type inference capabilities, we can spend less time having to write out things the compiler already knows. As all the types are deduced in the compiler phase only, the time for compilation increases slightly but it does not affect the run time of the program.

1) auto keyword: 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. Good use of auto is to avoid long initializations when creating iterators for containers.

> ***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.*

CPP

```
// C++ program to demonstrate working of auto
// and type inference

#include <bits/stdc++.h>
using namespace std;
```

```
    // 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
```

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++

```
// C++ program to demonstrate that we can use auto to
// save time when creating iterators
```

```
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();*

lets you extract the type from the variable so decltype is sort of an operator that evaluates the type of passed expression.

Explanation of the above keywords and their uses is given below:

CPP

```cpp
// C++ program to demonstrate use of decltype
#include <bits/stdc++.h>
using namespace std;

int fun1() { return 10; }
char fun2() { return 'g'; }

int main()
{
    // Data type of x is same as return type of fun1()
    // and type of y is same as return type of fun2()
    decltype(fun1()) x;
    decltype(fun2()) y;

    cout << typeid(x).name() << endl;
    cout << typeid(y).name() << endl;

    return 0;
}
```

Output

```
i
c
```

Below is one more example to demonstrate the use of **decltype**,

---

CPP

```cpp
// C++ program to demonstrate use of decltype
#include <bits/stdc++.h>
using namespace std;

// Driver Code
int main()
{
    int x = 5;
```

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

    return 0;
}
```

Output

```
 i
```

A program that demonstrates use of both auto and decltype

Below is a C++ template function **min_type()** that returns the minimum of two numbers. The two numbers can be of any integral type. The return type is determined using the type of minimum of two.

CPP

```
// C++ program to demonstrate use of decltype in functions
#include <bits/stdc++.h>
using namespace std;

// A generic function which finds minimum of two values
// return type is type of variable which is minimum
template <class A, class B>
auto findMin(A a, B b) -> decltype(a < b ? a : b)
{
    return (a < b) ? a : b;
}

// driver function to test various inference
int main()
{
    // This call returns 3.44 of double type
    cout << findMin(4, 3.44) << endl;

    // This call returns 3 of double type
    cout << findMin(5.4, 3) << endl;

    return 0;
}
```

Output

decltype vs typeid

- Decltype gives the type information at compile time while typeid gives at runtime.
- So, if we have a 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).

This article is contributed by **Utkarsh Trivedi**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Next

Range-based for loop in C++

Related Articles

1. Difference between Type Casting and Type Conversion
2. How to fix auto keyword error in Dev-C++
3. What is return type of getchar(), fgetc() and getc() ?
4. Data type of character constants in C and C++
5. Type Difference of Character Literals in C and C++
6. Function Overloading and Return Type in C++

Start Your Coding Journey Now!

8. Data Type Ranges and their macros in C++
9. Conversion of Struct data type to Hex String and vice versa
10. C | Storage Classes and Type Qualifiers | Question 1

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Easy

Easy | Normal | Medium | Hard | Expert

Improved By : Avesh_Agrawal, KundanSingh4, prakhar_11704180, anshikajain26, surinderdawra388

Article Tags : cpp-data-types, STL, C Language, C++

Practice Tags : CPP, STL

Improve Article | Report Issue

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Start Your Coding Journey Now!