

## Related Articles

# Type Inference in C++ (auto and decltype)

Difficulty Level : Easy • Last Updated : 05 Jan, 2021

Type Inference refers to automatic deduction of the data type of an expression in a programming language. Before C++ 11, each data type needs 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 compiler already knows. 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 keyword

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

---

## CPP


```
// 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

We have used typeid for getting the type of the variables. typeid is an operator which is used where dynamic type of an object need to be known. typeid(x).name() return shorthand name of the data type of x, for example, it return i for integers, d for doubles, Pi for the pointer to integer etc. But actual name returned is mostly compiler dependent. You can read more about typeid [here](#).

A 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
#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.

Explanation of above keyword and their uses is given below:

---

## 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

```
// 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
```

### A program that demonstrates use of both auto and decltype.

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

```
3.44
```

```
3
```

**decltype vs typeid:** `decltype` 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)).

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

Attention reader! Don't stop learning now. Get hold of all the important DSA concepts with the [DSA Self Paced Course](#) at a student-friendly price and become industry ready.



## RECOMMENDED ARTICLES

Page : [1](#) [2](#) [3](#)

### 01 Difference between Type Casting and Type Conversion

20, Apr 20

### 02 How to fix auto keyword error in Dev-C++

15, Jul 20

### 03 What is return type of getchar(), fgetc() and getc() ?

25, Feb 10

### 05 Type difference of character literals in C and C++

11, Nov 10

### 06 Function overloading and return type

05, Jan 11

### 07 Data Type Ranges and their macros in C++

05, Jul 16