TYPE INFERENCE IN C++

Type Inference is a feature in C++, using which the compiler automatically deduces the data type of an expression, function, or variable. Type inference was introduced with C++11 through the use of the auto and decltype.

Before C++ 11, each data type had to be explicitly declared, which limited the values of an expression at runtime. With type inference, all the data types are deduced by the compiler in the compiler phase only, due to which the time for compilation increases slightly, but it does not affect the run time of the program and greatly simplifies your code and improves readability.

auto in C++

The **auto** keyword in C++ is used in variable declarations or function declarations. It 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 compile time. The auto is commonly used to avoid long initializations when creating iterators for containers, or variables with complex data types.

Implement and run the following in a new project, within a file titled **Auto.cpp**

```
#include <iostream>
#include <typeinfo>
using namespace std;

int main() {

    //Using auto to avoid explicitly
    //defining the data type
    auto x = 4;
    auto y = 3.37;
    auto z = 3.37f;
    auto c = 'a';
    auto ptr = &x;

    //pointer to a pointer
    auto pptr = &ptr;
    cout << typeid(x).name() << endl
         << typeid(y).name() << endl
         << typeid(z).name() << endl
         << typeid(c).name() << endl
         << typeid(ptr).name() << endl
           << typeid(pptr).name() << endl;

    return 0;
}
```

In relation to the **Auto.cpp** program, note the following:

- A variable declared with auto keyword should be initialized at the time of its declaration only or else there will be a compile-time error.
- **typeid** has been used for getting the type of the variables.
    - **typeid** is an operator that is used where the dynamic type of an object needs to be known. Make it a point to find out more about this operator and **#include <typeinfo>**
- **typeid(x).name()** returns the data type of **x**, for example, it returns:
    - ‘i' for integers, 'd' for doubles,
    - 'f' for float, 'c' for char,
    - ‘Pi' for the pointer to the integer,
    - 'PPi' for the pointer to pointer to integer.
    - Single Pointer results in prefix 'P',
    - double-pointer results in 'PP' as a prefix and so on

decltype in C++

The **decltype** keyword in C++ inspects the declared type of an entity or the type of an expression. '**auto**' lets you declare a variable with a 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.

Implement and run the following in a new project, within a file titled **Decltype.cpp**.

```
#include <iostream>
#include <typeinfo>
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;

    int z = 5;
    // j will be of type int : data type of z
    decltype(z) j = z + 5;

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

    return 0;
}
```

Following are some main difference between **decltype** and **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).

References

- https://www.geeksforgeeks.org/cpp/type-inference-in-c-auto-and-decltype/