

# سلسلة تعلم البرمجة بلغة C++ الحديثة

Learn Modern C++ Programming Course

إعداد المهندس أحمد الديب



# **#5: Automatic Type Deduction**

# Type Aliases

```
typedef unsigned int* Pint32_t; // old  
using Pint32_t = unsigned int*;
```

# Lifetimes of Objects

- **Automatic:** Unless the programmer specifies otherwise, an object declared in a function is **created when its definition is encountered and destroyed when its name goes out of scope**. (allocated on the stack)
- **Static:** Objects declared in global or namespace scope and statics declared in functions or classes are **created and initialized once (only) and “live” until the program terminates**. (require locking to avoid data races)
- **Free store:** Using the **new and delete** operators, we can create objects whose lifetimes are controlled directly.
- **Temporary objects:** e.g. copy initialization as in case of pass by value.
- **Thread-local objects**

# auto

- `auto` for deducing a type of an object from its initializer.

```
int a1{123};  
char a2 = 123;
```

```
auto a3 = 123; // the type of a3 is int  
auto a4; // error: declaration of 'auto a4' has no initializer  
auto a5{123};
```

```
auto x1 = {1, 2}; // x1 type is std::initializer_list<int>  
auto x2 = {3}; // x2 type is is std::initializer_list<int>
```

```
auto x3 = {1, 2.0}; // error: cannot deduce element type  
auto x4{1, 2}; // error: not a single element
```

# auto Example

- There is not much advantage in using auto instead of int for an expression as simple as 123. The harder the type is to write and the harder the type is to know, the more useful auto becomes.

```
template <class T>
void f1(std::vector<T>& arg) {
    for (typename std::vector<T>::iterator p = arg.begin(); p != arg.end(); ++p)
        *p = 7;
}
```

```
template <class T>
void f2(std::vector<T>& arg) {
    for (auto p = arg.begin(); p != arg.end(); ++p) *p = 7;
}
```

# decltype()

- `decltype(expr)` for deducing the type of something that is not a simple initializer, such as the return type for a function or the type of a class member.

```
struct A {  
    double x;  
};  
const A a{0};  
  
decltype(a.x) y;  
std::cout << "y type is " << typeid(y).name() << std::endl;
```

# decltype() Example

```
// Generic Programming
template <typename T, typename U>
auto add(T t, U u) -> decltype(t + u) // suffix return type syntax
{
    return t + u;
}

int main() {
    auto result1{add(5, 4.5)};
    std::cout << "result1 type is " << typeid(result1).name() << std::endl;

    auto result2{add(5, 5)};
    std::cout << "result2 type is " << typeid(result2).name() << std::endl;
}
```

result1 type is d  
result2 type is i



**Thank you**