C++14 Language New Features Cheatsheet

Function return type deduction

```cpp
auto f_cpp14() { return 1; } // return type deduced to int

auto f_cpp11() -> int { return 1; } // trailing return type is needed in c++11
```

Variable templates

```cpp
template<typename T>
constexpr T pi = T(3.1415926535897932384626433833);

template<>
constexpr const char* pi<const char*> = "pi";

cout << pi<long double> << endl;
cout << pi<double> << endl;
cout << pi<float> << endl;
cout << pi<const char*> << endl;
```

Aggreagte member initialization

```cpp
struct foo_2 {
  int x{10}, y, z;
};
struct foo_2 f2 { 1, 2 }; // error in c++11

struct foo_3 {
  int x{1}, y{2}, z{3};
};
struct foo_3 f3_a { 1, 2 }; // error in c++11
```

Binary literals

```cpp
short i = 0b0101010101010101; // 0b
short j = 0B1101010101010101; // or 0B
```

The attribute deprecated

```cpp
class foo
{
public:
  // with message
  [[deprecated("x is not protected. Use getter instead")]]
  int x;
};
// without message
[[deprecated]] void f() {};
[[deprecated("g will not be supported from next release")]] void g() {};
```

Digit separator

```cpp
int i = 1'928'229'292; // single quote placed arbitrarily
int j = 1928'2'292'92; // for integer and floating point literals
```

Generic lambda

```cpp
auto lambda = [](auto x, auto y) { return x + y; };

cout << lambda(1, 2) << endl; // x and y deduced to int
cout << lambda(1.6, 2.5) << endl; // x and y deduced to float
string s1{"1"}, s2{".5"};
cout << lambda(s1, s2) << endl; // x and y deduced to std::string
```

Relaxed constexpr restrictions

constexpr functions may contain:

- declarations (except *static* or *thread_local*)
- *if* and *switch*
- loops

```cpp
constexpr int f_cpp14(int x) {
  if (x % 2 == 0)
    return x * 10;
  return x;
}
```

Lambda capture expression

```cpp
auto lambda_constant = [value = 3](int x) { return value * x; };

auto lambda_func_call = [value = f(argc)](int x) { return value * x; };

string s{"foo"};
auto lambda_move = [value = move(s)](int x) { return value + to_string(x); };
```

Alternate type deduction on declaration

```cpp
int i;
int&& f();
auto x3a = i;                  // decltype(x3a) is int
decltype(auto) x3d = i;        // decltype(x3d) is int
auto x4a = (i);                // decltype(x4a) is int
decltype(auto) x4d = (i);      // decltype(x4d) is int&
auto x5a = f();                // decltype(x5a) is int
decltype(auto) x5d = f();      // decltype(x5d) is int&&
auto x6a = { 1, 2 };           // decltype(x6a) is std::initializer_list<int>
decltype(auto) x6d = { 1, 2 }; // error, { 1, 2 } is not an expression
auto *x7a = &i;                // decltype(x7a) is int*
decltype(auto)*x7d = &i;   // error, declared type is not plain decltype(auto)
```