C++11 Knowledge Point

* Chapter 2
  + 2.4.1. References to const
  1. Terminology: const Reference is a Reference to const.Technically speaking, there are no const references.
  + 2.4.2. Pointers to const
  1. It may be helpful to think of pointers and references to const as pointers or references “that think they point or refer to const.”
  2. Like a references to const
  3. const pointers: pointer that is itself const.the address that it holds can not be changed.
  + 2.4.3. Top-Level const
  1. top-level const to indicate that the pointer itself is a const.
  2. pointer can point to a const object，we refer to that const as a low-level const.
  + 2.4.4. constexpr and Constant Expressions
  1. constant expression is an expression whose value cannot change and that can be evaluated at compile time.
  2. Literal Types:the arithmetic, reference, and pointer types are literal types
  3. Variables defined inside a function ordinarily are not stored at a fixed address Hence,we cannot use a constexpr pointer to point to such variables.On the other hand, the address of an object defined outside of any function is a constant expression.
  4. Pointers and constexpr: constexpr specifier applies to the pointer.

e.g. const int \*p = nullptr; constexpr int \*q = nullptr;

p is a pointer to const int,q is a const pointer to int.

* + 2.5.1. Type Aliases
  1. two ways:

typedef e.g.

typedef double wages; // wages is a synonym for double

typedef wages base, \*p; // base is a synonym for double, p for double\*

using e.g.

using SI = Sales\_item; // SI is a synonym for Sales\_item

* 1. Pointers, const, and Type Aliases

typedef char \*pstring;

const pstring cstr = 0; // cstr is a constant pointer to char

const pstring \*ps; // ps is a pointer to a constant pointer to char

The base type in these declarations is const pstring.

* + 2.5.2. The auto Type Specifier
    - 1. By implication ,a variable that uses auto as its type specifier must have an initializer.
      2. When we define multiple variables using auto,initializers for all the variables in the declaration must have types that are consistent with each other.
      3. The compiler adjusts the type to conform to normal initialization rules:

First: when we use reference,we are really use the object to which the reference refers.

Second: auto ordinarily ignores top-level consts; low-level consts, such as when an initializer is a pointer to const, are kept.

Third: if we want the deduced type to have a top-level const, we must say so explicitly.

e.g. const auto f = ci;// deduced type of ci is int; f has type const int

Forth: When we ask for a reference to an auto-deduced type, top-level consts in the initializer are not ignored. As usual, consts are not top-level when we bind areference to an initializer.

* + 2.5.3. The decltype Type Specifier
  1. returns the type of its operand. The compiler analyzes the expression to determine its type but does not evaluate the expression.
  2. The way decltype handles top-level const and references differs subtly from the way auto does.

When the expression to which we apply decltype is a variable, decltype returns the type of that variable, including top-level const and references.

* 1. decltype and References

When we apply decltype to an expression that is not a variable, we get the type that that expression yields.Generally speaking, decltype returns a reference type for expressions that yield objects that can stand on the left-hand side of the assignment

* 1. decltype of a parenthesized variable is always a reference(diff with auto)
  + 2.6.1. Defining the Sales\_data Type