【C++】Day18

Class	C++
Date	@December 8, 2021
Material	
Series Number	
Summary	

【Ch6】Functions

6.4 Overloaded Functions

Functions that have the same name but different parameter lists and that appear in the same scope are overloaded.

For example, we define several functions named print:

```
void print(const char *cp);
void print(const int *beg, const int *end);
void print(const int ia[], size_t size);
```

When we call these functions, the compiler can deduce which function we want based on the argument type we pass:

```
int j[2] = {0, 1};
print("Hello"); //call print(const char *cp)
print(j, end(j) - begin(j)); //call print(const int ia[], size_t size)
print(begin(j), end(j)); //call print(const int *beg, const int *end)
```

Note: The main function may not be overloaded.

It is an error for two functions to differ only in terms of their return types. If the parameter lists of two functions match but the return types differ, the second declaration is an error:

```
Record lookup(const Account&);
bool lookup(const Account&); //error: only the return type is different
```

Determining Whether Two Parameter Types Differ

Two parameter lists can be identical, even if they don't look the same:

```
Record lookup(const Account &acc);
Record lookup(const Account&); //parameter names are ignored

typedef Phone Telno;
Record lookup(const Phone&);
Record lookup(const Telno&); //Telno and Phone are the same type
```

In the second pair, it looks like the types are different, but Telno is not a new type; it is a synonym for Phone. A type alias provides an alternative name for an existing type; it does not create a new type.

Overloading and const Parameters

Top-level const has no effect on the objects that can be passed to the function. A parameter that has a top-level const is indistinguishable from one without a top-level const:

```
Record lookup(Phone);
Record lookup(const Phone); //redeclares Record lookup

Record lookup(Phone*);
Record lookup(Phone *const);
```

On the other hand, we can overload based on whether the parameter is a reference to the const or nonconst version of a given type, such consts are low-level

```
Record lookup(Phone*);
Record lookup(const Phone*);
```

```
Record lookup(Account&);
Record lookup(const Account&);
```

6.4.1 Overloading and Scope

Warning: Ordinarily, it is a bad idea to declare a function locally.

Programmers new to C++ are often confused about the interaction between scope and overloading. However, overloading has no special properties with respect to scope: As usualy, if we declare a name in an inner scope, that name hides uses of that name declared in an outer scope. Names do not overload across scopes:

```
void print(const string &s);
string read();

void fooBar(int ival) {
  bool read = false; //new scope: hides the outer declaration of read
  void print(int); //new scope: hides previous instances of print
}
```

Note: In C++, name lookup happens before type checking.