【C++】Day five

Class	C++
Date	@November 22, 2021
Material	
Series Number	
Summary	

【Ch2】Identifiers

2.2.3 Identifiers

Identifiers in C++ can be composed of letters, digits, and the underscore character.

Identifiers must begin with either a letter or an underscore. Identifiers are case-sensitive, which means upper- and lowercase letters are distinct

The language reserves a set of names that cannot be used as identifiers.

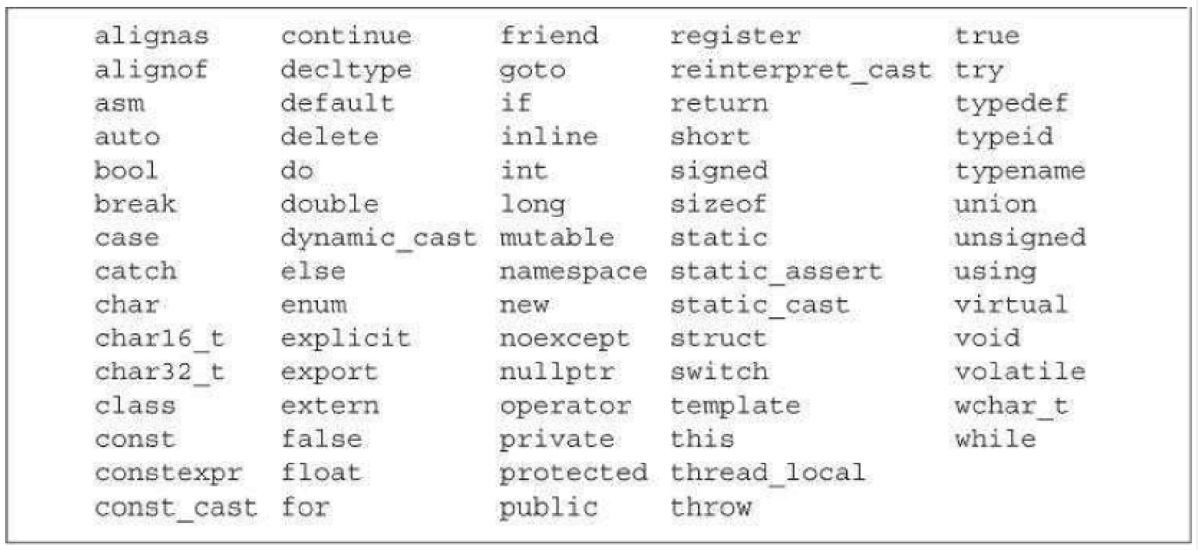
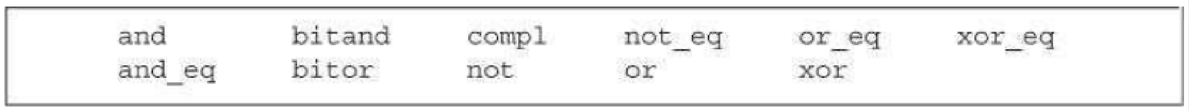


Table 2.3. C++ Keywords

alignas	continue	friend	register	true
alignof	decltype	goto	reinterpret_cast	try
asm	default	if	return	typedef
auto	delete	inline	short	typeid
bool	do	int	signed	typename
break	double	long	sizeof	union
case	dynamic_cast	mutable	static	unsigned
catch	else	namespace	static_assert	using
char	enum	new	static_cast	virtual
char16_t	explicit	noexcept	struct	void
char32_t	export	nullptr	switch	volatile
class	extern	operator	template	wchar_t
const	false	private	this	while
constexpr	float	protected	thread_local	
const_cast	for	public	throw	

Table 2.4. C++ Alternative Operator Names



and	bitand	compl	not_eq	or_eq	xor_eq
and_eq	bitor	not	or	xor	

The standard also reserves a set of names for use in the standard library. The identifiers we define in our own programs may not contain two consecutive underscores, not can an identifier begin with an underscore followed immediately by an uppercase letter.

Identifiers defined outside a function may not begin with an underscore.

Several conventions that improve the readability of a program:

- Variable names normally are lowercase. For example, `int index = 1;`
- Classes we define usually begin with an uppercase letter. For example, `Sales_item`
- Identifiers with multiple words should visually distinguish each word. For example, `student_loan` or `studentLoan`.

2.3 Compound Types

A compound type is a type that is defined in terms of another type.

2.3.1 References

A reference defines an alternative name for an object. A reference type "refers to" another type.

We define a reference type by writing a declarator of the form `&d`, where `d` is the name being declared.

```
int ival = 1024;
int &refVal = ival; //refVal refers to (is another name for) ival
int &refVal2; //error: a reference must be initialized
```

When we initialize a variable, the value of the initializer is copied into the object we are creating.

When we define a reference, instead of copying the initializer's value, we bind the reference to its initializer.

Once initialized, a reference remains bound to its initial object. There is no way to rebind a reference to refer to a different object. Because there is no way to rebind a reference, references must be initialized.

Note: A reference is not an object. Instead, a reference is just another name for an already existing object.

After a reference has been defined, all operations on that reference are actually operations on the object to which the reference is bound:

```
refVal = 2; //assign 2 to the object to which refVal refers
int ii = refVal; //same as ii = iVal
```

- When we assign to a reference, we are assigning to the object to which the reference is bound.
- When we fetch the value of a reference, we are really fetching the value of the object to which the reference is bound.
- When we use a reference as an initializer, we are really using the object to which the reference is bound

```
int &refVal3 = refVal; //ok: refVal3 is bound to the object to which refVal is bound
int i = refVal; //ok: initializes i to the same value as iVal
```

Reference Definition

We can define multiple references in a single definition. Each identifier that is a reference must be preceded by the `&` symbol.

```
int a = 30, &refVal = a; //a is an int, refVal is a reference bound to a
```

The type of a reference and the object to which the reference refers must match exactly.

A reference must be bound only to an object, not to a literal or to the result of a more general expression:

```
int &refVal = 10; //error: initializer must be an object
double dVal = 3.14;
int &refVal2 = dVal; //error: initializer must be an int type.
```