【C++】Day four

Class	C++
Date	@November 21, 2021
Material	
Series Number	
Summary	

【Ch2】Variables

A variable provides us with named storage that our programs can manipulate. Each variable in C++ has a type.

The type determines the size and layout of the variable's memory, the range of values that can be stored within that memory, and the set of operations that can be applied to the variable.

2.2.1 Variable Definitions

A simple variable definition consists of a type specifier, followed by a list of one of more variable names separated by commas, and ends with a semicolon.

A definition may optionally provide an initial value for one or more of the names it defines.

```
int a = 3;
```

List Initialization

The language defines several ways to initialize a variable:

```
int units_sold = 0;
int units_sold = {0};
```

```
int units_sold{0};
int units_sold(0);
```

Braced lists of initializers can now be used whenever we initialize an object and in some cases when we assign a new value to an object.

When used with variables of built-in type, this form of initialization has one important property: The compiler will not let us list initialize variables of built-in type if the initializer might lead to the loss of information:

```
long double ld = 3.1415;
int a{ld}, b = {ld}; //error: narrowing conversion required
int c(ld), d = ld; //ok: but values will be truncated
```

Default Initialization

When we define a variable without an initializer, the variable is default initialized. Such variables are given the "default" value. What the default value is depends on the type of the variable and may also depend on where the variable is defined.

Variables defined outside any function body are initialized to zero.

The value of an uninitialized variable defined inside a function of built-in type is undefined, and it is an error to copy or otherwise try to access the value of a variable whose value is undefined.

Each class controls how we initialize objects of that class type. It is up to the class whether we can define objects of that type without an initializer.

Most classes let us define objects without explicit initializers. Such classes supply an appropriate default value for us.

If the string class says that if we do not supply an initializer, the the resulting string is the empty string:

```
std::string empty; //empty implicityly initialized to the empty string
```

Some classes require that every object be explicitly initialized.

2.2. Variable Declarations and Definitions

To support separate compilation, C++ distinguishes between declarations and definitions. A declaration makes a name known to the program. A file that wants to use a name defined elsewhere includes a declaration for that name.

A definition creates the associated entity.

To obtain a declaration that is not also a definition, we add the extern keyword and may not provide an explicit initializer:

```
extern int i; //declares but does not define i
int j; //declares and defines j
```

An extern that has an initializer is a definition

```
extern double pi = 3.14; //definition
```

Static Typing

C++ is a statically typed language, which means that types are checked at compiile time. The compiler checks whether the operations we write are supported by the types we use. If we try to do things that the type does not support, the compiler generates an error message and does not produce an executable file.