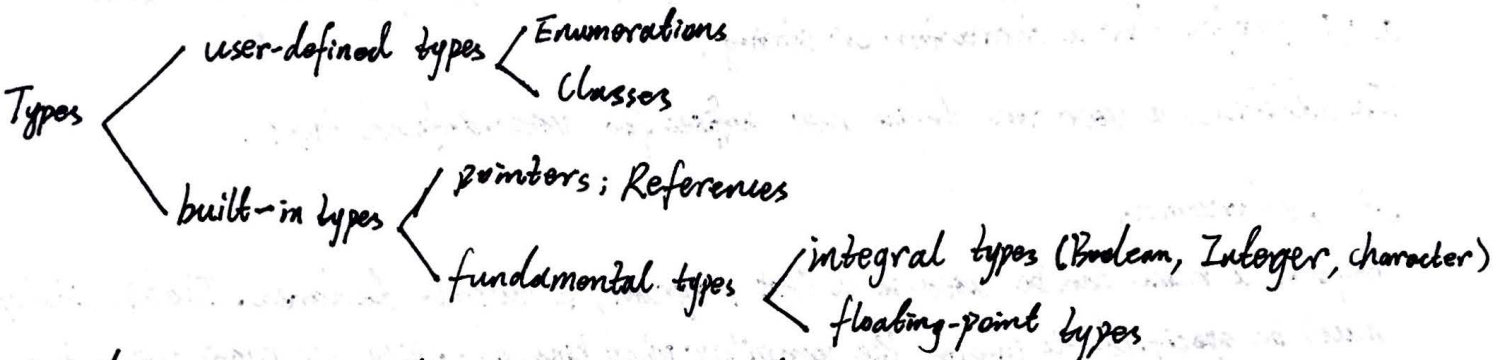
Types and Declarations

6.2.1 Fundamental Types

Boolean; Character; Integer; Floating-point; Void (used to signify the absence of info)

pointers; Arrays; References; Data structures and classes; enumerations.



A widening conversion changes a value to a data type that can allow for any possible value of the original data.

A narrowing conversion changes a value to a data type that might not be able to hold some of the possible values.

6.2.3.1 Signed and Unsigned Characters

It is implementation-defined whether a plain char is considered signed or unsigned.

* One solution is to avoid plain char and use the specified char types only.
* You can't mix pointer to different char types.

6.2.4 Integers

plain ints are always signed.

A literal starting with zero followed by x or X is a hexadecimal number.

A literal starting with zero ~~by~~ but not followed by x or X is an octal number.

plain 0 is octal not decimal

The suffix U can be used to write explicitly unsigned literals. The suffix L can be used to write explicitly long literals.

Combinations of suffixes are allowed.

6.2.5.1 Floating point numbers

If you want a floating point literal of type float, you can define using the suffix f or F

If you want a floating-point literal of type long double, you can define one using suffix l or L

6.2.6 Prefixes and Suffixes

Note the dramatic difference in the meaning of a U suffix for an integer and a U prefix for a character or string.

In addition, a user can define new suffixes for user-defined types.

6.3 Declarations

Before a name can be used in a C++ program, it must be declared. That is, its type must be specified to inform the compiler what kind of entity the name refers to.
A definition is a declaration that supplies all that is needed in a program for the use of an entity

```
using Point = std::complex<short>; // Point is a name for std::complex<short>
```

There must always be exactly one definition for each name in a C++ program. However, there can be many declarations. All declarations of an entity must agree on its type.

Any declaration that specifies a value is a definition.

Of the definitions, only two do not specify values:

```
char ch;
string s;
```

Operators apply to individual names only, not to any subsequent names in the same declaration

{ } list-initialization. An empty initializer list { } is used to indicate that a default value is desired.

prefer = when using auto

The harder the type is to write and the harder the type is to know, the more useful auto is

The type of an expression is never a reference because reference are implicitly dereferenced in expression.

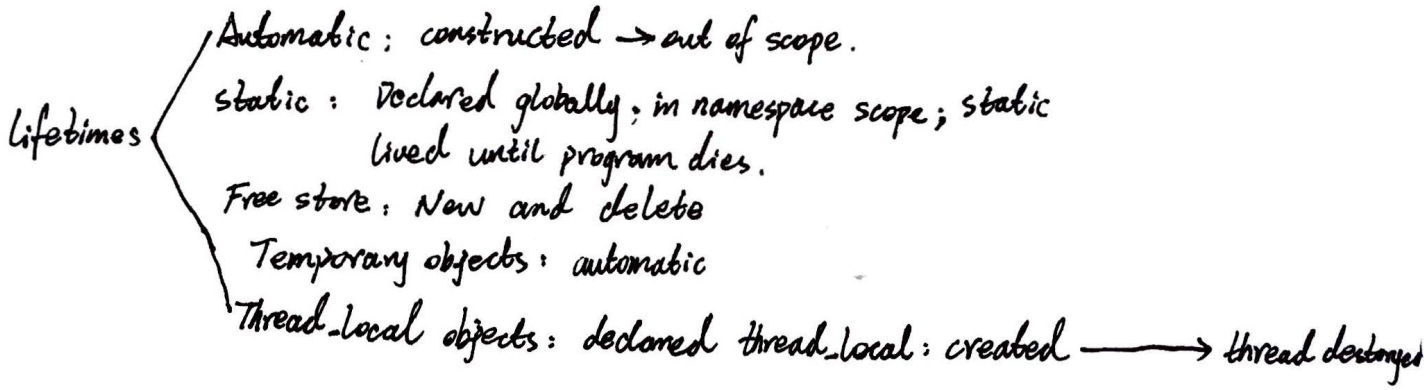
auto + '=' or use { }

An object is an contiguous region of storage;

An lvalue is an expression that refers to an object.

A classical lvalue is something that has identity and cannot be moved.

A classical rvalue is something/anything that we are allowed to move from.



6.5 Type Aliases

```
using value_type = T;
```

or `typedef value_type T;`

we cannot apply type specifiers such as unsigned to an alias.