1 Types

In C++, you can use the basic types to build more complex types of your own design. But you will never use it, because classes are far more powerful. Still, it is important to know about the following ways of building types so that you will recognize the syntax.

1.1 Enumerated Types

Enumerated types let you define your own sequences so that you can declare variables with values in that sequence. As an example, in a chess program, you *could* represent each piece as an int, with const to indicate that they can never change.

```
const int PieceTypeKing = 0;
const int PieceTypeQueen = 1;
const int PieceTypeRook = 2;
const int PieceTypePawn = 3;
//etc
int myPiece = PieceTypeKing;
```

However, this can become dangerous. Since a piece is just an int, another programmer could add code to increment the value of a piece. Enumerated types solve these problems by tightly defining the range of values for a variable. The following declares a new type, PieceType, representing chess pieces.

```
enum PieceType
{
  PieceTypeKing,
  PieceTypeQueen,
  PieceTypeRook,
  PieceTypePawn
};
```

By defining possible values for variables of type PieceType, the compiler would give a warning or error if you attempt to perform arithmetic on PieceType variables or treat them as integers. The following code results in a warning or error on most compilers.

```
   PieceType myPiece;
   myPiece = 0;
```

It is also possible to specify integer values for members of an enumeration. The syntax is as follows.

```
enum PieceType
{
  PieceTypeKing = 1,
  PieceTypeQueen,
  PieceTypeRook = 10,
  PieceTypePawn
};
```

In this case, PieceTypeQueen has the value 2, and PieceTypePawn has the value 11, automatically assigned by the compiler.

1.2 Strongly Typed Enumerations

Enumerations above are not strongly typed, meaning they are not type-safe. They are always interpreted as integers, and thus you can compare enumeration values from completely different enumeration types. The enum class solves these problems. For example:

```
enum class MyEnum
{
  EnumValue1,
  EnumValue2 = 10,
  EnumValue3
};
```

This is a type-safe enumeration called MyEnum. These enumeration value names are not automatically exported to the enclosing scope, which means you always have to use the scope resolution operator:

```
MyEnum value1 = MyEnum::EnumValue1;
```

The enumeration values are not automatically converted to integers, which means the following is illegal

```
if (MyEnum::EnumValues3 == 11){...}
```

By default, the underlying type of an enumeration is an integer, but this can be changed as follows:

```
enum class MyEnumLong : unsigned long
{
   EnumValueLong1,
   EnumValueLong2 = 10,
   EnumValueLong3
};
```