32. Enumerations

Objectives

- Create enums
- Use enums
- Understand the relationship between enums and constants

Up to this point we have been using types provided by C++. Using these types we have also been using arrays and pointer types. Now we will create our own types using `enum`. The `enum` basically creates a type made up of integer constants.

Enum

There's another way to rewrite a previous example:

```
typedef int EmployeeCode;

const EmployeeCode CEO = 0;
const EmployeeCode MANAGER = 1;
const EmployeeCode FULLTIME = 2;
const EmployeeCode PARTTIME = 3;

EmployeeCode code = CEO;
```

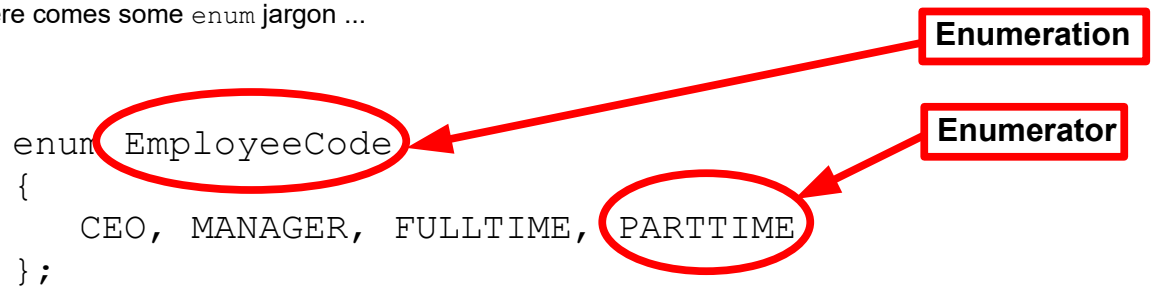
Here's how you do it:

```
#include <iostream>

int main()
{
    enum EmployeeCode
    {
        CEO, MANAGER, FULLTIME, PARTTIME
    };

    EmployeeCode code = CEO;
    std::cout << CEO << std::endl;
    std::cout << code << std::endl;
    return 0;
}
```

Here comes some `enum` jargon ...



The format of creating an enumeration is

```
enum [enumeration name]
{
    [enumerator-1], ...,
    [enumerator-n]
};
```

Exercise. Declare a `Grade` enumeration containing enumerators `A`, `B`, `C`,

`D`, and `F`. Declare variable `johnDoeGrade` of type `Grade` and initialize it with `A`. Print `johnDoeGrade` and then print `A`.

Enum as a type

So what's the difference between

```
typedef int EmployeeCode;

const EmployeeCode CEO = 0;
const EmployeeCode MANAGER = 1;
const EmployeeCode FULLTIME = 2;
const EmployeeCode PARTTIME = 3;
```

and

```
enum EmployeeCode
{
     CEO, MANAGER, FULLTIME, PARTTIME
};
```

(besides the fact that the second produces a shorter code)???

The enumeration `EmployeeCode` is actually ***a new type***. This means that you can actually have this:

Exercise. Try compiling this program:

```
enum EmployeeCode
{
    CEO, MANAGER, FULLTIME, PARTTIME
};

void f(EmployeeCode c)
{}

void f(int i)
{}

int main()
{
    return 0;
}
```

Enumerators and Integers

You might say ... “Hang on there! What do you mean that the `EmployeeCode` is a new type??? When I print an `EmployeeCode` variable like this ...”

```
enum EmployeeCode
{
    CEO, MANAGER, FULLTIME, PARTTIME
};
std::cout << CEO << std::endl;
```

“I see that `CEO` *is* 0!!! So it *is* an integer!!! Isn't it???”

Yes and no.

There's actually an automatic *typecasting* between enumerators and integers. The statements:

```
std::cout << CEO << std::endl;
std::cout << MANAGER << std::endl;
```

are really the same as

```
std::cout << int(CEO) << std::endl;
std::cout << int(MANAGER) << std::endl;
```

This means that `CEO`, because it's the ***first*** enumerator of an enumeration, is automatically typecast to the integer ***0*** if necessary, `MANAGER` is automatically typecast to the integer 1 if necessary, etc.

That example shows you that `EmployeeCode` enumerator is automatically typecast to an `int` when necessary. What about from an `int` to `EmployeeCode` enumerator? Try this:

```
enum EmployeeCode
{
    CEO, MANAGER, FULLTIME, PARTTIME
};

EmployeeCode code = EmployeeCode(1);
```

The last statement is the same as:

```
EmployeeCode code = MANAGER;
```

Now let's see if the typecast is also automatic. Try this:

```
enum EmployeeCode
{
    CEO, MANAGER, FULLTIME, PARTTIME
};

EmployeeCode code = 1; // 1 becomes EmployeeCode(1)??
```

The point: An integer is ***not*** automatically typecast to an enumerator.

Since C/C++ will automatically typecast enumerators to integers if necessary, we can use it as though it's an integer. Here are some examples:

Run this:

```
enum operation {ADD, SUB, MUL, DIV};
operation op = ADD;
int x = 10, y = 20;

switch (op)
{
    case ADD: std::cout << x + y; break;
    case SUB: std::cout << x - y; break;
    case MUL: std::cout << x * y; break;
    case DIV: std::cout << x / y; break;
}
```

Try changing the `op` to different values.

Exercise. Is this a valid code segment? If so, what is the output? If it isn't explain why.

```
enum operation {ADD, SUB, MUL, DIV};
std::cout << ADD + SUB + MUL + DIV << std::endl;
```

The integer value of an enumerator

You can actually change the integer value associated with an enumerator. Try this:

```
enum CutOff
{
    A = 90, B = 80, C = 70, D = 60, F = 0
};

std::cout << B << std::endl;

CutOff x = B;
std::cout << x << std::endl;
```

Now try this:

```
enum X
{
  a = 5, b, c, d = 42, e, f
};
std::cout << a << ' ' << b << ' ' << c << ' '
          << d << ' ' << e << ' ' << f << std::endl;
```

And ... you ***have*** to try this because it's a common gotcha:

```
enum X {a = 1.1};
```

The point is that enumerators can only be initialized with integer values.

Exercise. Either write down the output of the following code segment or find the error:

```
enum X
{
    a, b, c, d = 42, e, f
};
X foo = b;
X bar = f;
std::cout << b << f << std::endl;
```

Exercise. Either write down the output of the following code segment or find the error:

```
enum Y
{
    a = 1, b, c, d = 4.5, e = 5.6, f = 6.7
};
X foo = b;
std::cout << b << std::endl;
```

Enum Without Enumeration Name

Notice that

```
enum EmployeeCode
{
    CEO, MANAGER, FULLTIME, PARTTIME
};
```

actually does ***two*** things:

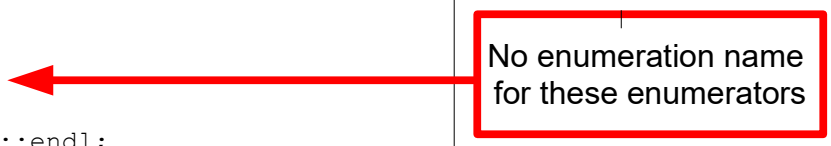
- It creates four constants (enumerators)
- It gives a type (enumeration) for the four constants.

You can actually use the `enum` to create constants without creating an enumeration. Try this:

```
#include <iostream>

int main()
{
    enum {A, B, C, D, E};

    std::cout << A << std::endl;
    return 0;
}
```



Exercise. Rewrite the following using enumerators:

```
const int FEE = 1;
const int FIE = 0;
const int FOE = 2;
const int FUM = 3;
```

Omit the enumeration name and integer values.

Symbolic constants

Sometimes we don't really care about the integer value of enumerators. We really want the names of the enumerators to make it easier to read our programs. Look at this example:

```
enum GameState
{
    INIT, MENU, STARTING, RUN, RESTART, EXIT
};

GameState gameState = INIT;

int main()
{
    int energy = 1000;

    while (gameState != EXIT)
    {
        ...
        if (energy == 0)
            gameState = EXIT
        ...

        switch (gameState)
        {
            case INIT: ...
            case MENU: ...
            case STARTING: ...
            case RUN: ...
            case RESTART: ...
            case EXIT: ...
        }
    }

    return 0;
}
```

Notice that it doesn't matter if the integer value of `INIT` is 0 or 1 or 2 or what-have-you.

Using this technique of programming where the code to execute is based on the value of a variable, you create something called a finite state machine.

Exercise. Given

```
enum EnumA {x, y, z};
enum EnumB {a, b, c};
```

does this work?

```
EnumA i = a;
```

Exercise. Given

```
enum EnumA {x, y, z};
enum EnumB (a, b, c};
```

does this work?

```
EnumA a0 = x;
EnumB a1 = a;
std::cout << (a0 == a1) << std::endl;
```

Multi-file Compilation

Enumerations can be placed in a header file if other files (source or header) use it.

Summary

You can create symbolic constants using enum.

You can give the set of constants a name. This creates a type.

You can then create a variable of an enum type.

```
enum PlaneState { STATIONARY, FLYING, MAINTENANCE };
PlaneState boeing747a = STATIONARY;
PlaneState boeing747b = FLYING;
```

By default enumerators are given integer values: 0, 1, 2, ...

You can give the enumerators values:

```
enum PlaneState
{
  STATIONARY=5, FLYING, MAINTENANCE=10
}; // int(FLYING) is 6
```

The name of the enumeration is optional.