# 05. Constants

**Objectives**

- Choose appropriate names for constants.
- Declare constants
- Use constants in expressions

This is a pretty short section. We want to talk about constants. A constant is just a variable whose value cannot be changed once it's initialized.

# Constants

## A **constant** is just a variable whose **value cannot change**.

## A constant must be initialized.

This example will show you how to create constant:

```
const double PI = 3.14159;
```

**Exercise.** What if we don't initialize but assign after declaration? Does it work?

```
const double PI;
PI = 3.14159;
```

**Exercise.** What if we assign after the initialization? Does it work?

```
const double PI = 3.14159;
PI = 3.14159265;
```

The format for the declaration of a constant in C++ looks like this:

```
const [type] [var name] = [value];
```

Of course the `[value]` can be an expression that can evaluate to a value. The expression must be a **constant expression**. This means that the expression cannot have a nonconstant variable. Try these example:

**Exercise.** Try this:

```
const int a = 1;
const int b = a;
const int c = a + 2 * b + 3;
std::cout << a << ' ' << b << ' ' << c << '\n';
```

**Exercise.** Now try this:

```
int a = 1;
const int b = a;
```

The standard practice for naming constants is that you use uppercase letters and you separate words in the variable name with the underscore. Here are some examples:

```
const double MAX = 41.41;
```

```
const double MAX_POWER = 123.41;
const int MAX_LIVES = 3;
const int MONTHS_PER_YEAR = 12;
const int HOURS_PER_WEEK = 40;
```

The only exception is when you're writing scientific applications where there are lots of constants and their names have already been standardized for a long time. Remember that a programmer always strives to write code that's easy to read. For instance

```
const double e = 2.718; // GOOD
```

and not

```
const double E = 2.718; // BAD, BAD, BAD!!!
```

Remember that a constant is a variable. Therefore you can print it and you can use all the relevant operators on it. The only operators you cannot use are those that attempt to change the value of the constant (I've already mentioned that you cannot use the assignment after a constant has been initialized.)

**Exercise.** Does this work?

```
const double x = 1.2345;
std::cin >> x;
```

(Duh)

# Why Constants?

Let's say you work in a company where the boss wants a program to compute the annual salary of an employee. First run this program:

```
int monthly_salary;
std::cout << "Enter salary per month: ";
std::cin >> monthly_salary;
std::cout << "In 12 months the employee earns $"
          << monthly_salary * 12 << std::endl;
```

Now suppose the boss decided to have 11 paid months. (Yes, he's a slave driver.) Modify the program. Run your program and make sure it works.

But he changed his mind. He's going to pay 10 months of salary. Change the program. (Grrrrr...) Make sure it works.

Now I want you to modify your program to get this:

```
const int PAID_MTHS_PER_YR = 10;

int monthly_salary;
std::cout << "Enter salary per month: ";
std::cin >> monthly_salary;
std::cout << "In " << PAID_MTHS_PER_YR
          << " months the employee earn $"
          << monthly_salary * PAID_MTHS_PER_YR
          << '\n';
```

After a few days, your boss felt real bad. He decided to pay everyone 12 months. Change your program accordingly.

There are at least two reasons why we use constants:

Constants give **meaningful names to values so that the code is more readable**. Plain-jane values have no meanings. They are just values. Constants make C++ code easier to read. For instance this is easier to comprehend:

```
std::cout << pi * r * r << std::endl;
```

than this:

```
std::cout << 3.1415926535897 * r * r << std::endl;
```

If the constant value needs to be changed, just change at one spot - in the declaration of the constant. Therefore using **constants make code easier to maintain in the event of changes.**

Of course you can create variables for the above two reasons too like

this:

```
const int PAID_MTHS_PER_YR = 10;

int monthly_salary;
std::cout << "Enter salary per month: ";
std::cin >> monthly_salary;
std::cout << "In " << PAID_MTHS_PER_YR
          << " months the employee earn $"
          << monthly_salary * PAID_MTHS_PER_YR
          << std::endl;
```

i.e., `PAID_MTHS_PER_YR` is not a constant.

Why do we want the value of the constant variable not to change? It's a safety measure. If one day you (or your colleague) accidentally assign a value to a constant, C++ will yell at you (or your colleague). Therefore

# constants prevent accidental changes to the value of a variable that should not be modified.

Writing values directly into the code instead of using constants is called "hard-coding constants".

Of course you only create constants to make a program easier to read and maintain. You don't create constants for every single value that can appear in your program! For instance don't change the following program:

```
std::cout << "triangle area:"
          << 0.5 * base * height << '\n';
```

to this:

```
const double HALF = 0.5;       // a silly constant!!!
std::cout << "triangle area:"
          << HALF * base * height << '\n';
```

This doesn't help! "Half" ***is*** 0.5!!!

**Exercise.** Write a program that computes the area of a circle when the user enters a value for the radius. There should be a constant.

**Exercise.** According to Elbert Ainstain the energy of a body of mass m is given by

$$E = mc^3$$

where c is the speed of light which is 123.45 meter per second. Write a program that prompts the user for m and prints E up to 4 decimal places. There should be a constant in your code.

**Exercise.** The CEO of VyHee has asked you to write program to print some salary data. Employees on hourly wages are paid at a fixed rate of

$123.89 per hour. Such employees are required to work 120 per week. All other employees are paid $4242 per month. There are 50 work weeks at VyHee. You need to write a program that prompts the user for relevant data and print the total amount that the CEO must pay for salaries. Include the fact the CEO himself/herself makes $1,000,000 per year. Use as many constants as you should.

# C-style constants

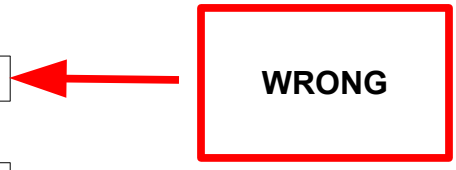
This is a DIY section.

There's another way to create constants in C++ programs. This is in fact from the C language. (C++ is a descendant of C and therefore this method of creating constants are carried forward to C++.) Run this:

```
#include <iostream>

#define NUM_ARMS 3
#define HELLO_WORLD "HELLO... WORLD... !!!"

int main()
{
    std::cout << NUM_ARMS << '\n'
              << HELLO_WORLD << '\n';

    return 0;
}
```

Note the syntax. There's no = sign next to NUM_ARMS:

```
#define NUM_ARMS = 3
```




You can do this if you like (but it's not so common):

```
#include <iostream>

int main()
{
    #define NUM_ARMS 3
    #define HELLO_WORLD "HELLO... WORLD... !!!"

    std::cout << NUM_ARMS << '\n'
              << HELLO_WORLD << '\n';

    return 0;
}
```

The `#define` is not part of the C or C++ language. Briefly, your MS VS .NET will actually replace NUM_ARMS with 3 even before the program is compiled. In other words before compiling (and running the program), there's a ***textual replacement*** of `NUM_ARMS` by `3`. This textual replacement process will actually take the above, and give you this:

```
#include <iostream>

int main()
{
    std::cout << 3 << '\n'
              << "HELLO... WORLD... !!!" << '\n';

    return 0;
}
```

and ***then*** … this above program is sent to the compiler.

As much as possible you should avoid using C-style constants.

You should be aware of this method of creating constants since
1. many C++ programmers (unfortunately) use it occasionally
2. you might one day inherit C legacy code.

The `#define` however can be used in another context. We will talk about that later.

# Summary

A constant is a variable whose value cannot change. The declaration of constants look like this:

```
const [type] [var name] = [value];
```

Constants must be initialized. You can initialize a constant with a value or an expression that involves constants. They cannot be assigned values after their initialization.

The name of a constant is usually in uppercase letters with words separated by the underscore. The only exception is when the constant it represents already has an established name.

# Exercises

Q1. Modify the following code by creating meaningful constants in place of integer values and variables with meaningful names:

```
int i = 0;
std::cout << "how many toyota priuses do u want?";
std::cin >> i;

int j = 0;
std::cout << "how many ipods do u want?";
std::cin >> j;

double k = 0;
std::cout << "how many more yrs in college?";
std::cin >> k;

double l = 0;
std::cout << "how much do u have?";
std::cin >> l;

double m = 0;
std::cout << "how much can u borrow from mom?";
std::cin >> m;

std::cout << "you need this much:"
          << l + m - 30000 * i - 290 * j - 15000 * k
          << '\n';
```

Q2. Declare the following constants for a tic-tac-toe program:

- A constant representing the number of rows in the tic-tac-toe board.
- A constant representing the number of columns in the tic-tac-toe board.
- A constant representing the number of squares in the tic-tac-toe board.

Print all the constants. The third constant should depend on the first two. This means that if there are changes to board size, you only need to change the first two constants.

Q3. List down all the benefits for using constants – don't look at the notes! Once you're done check against the notes.