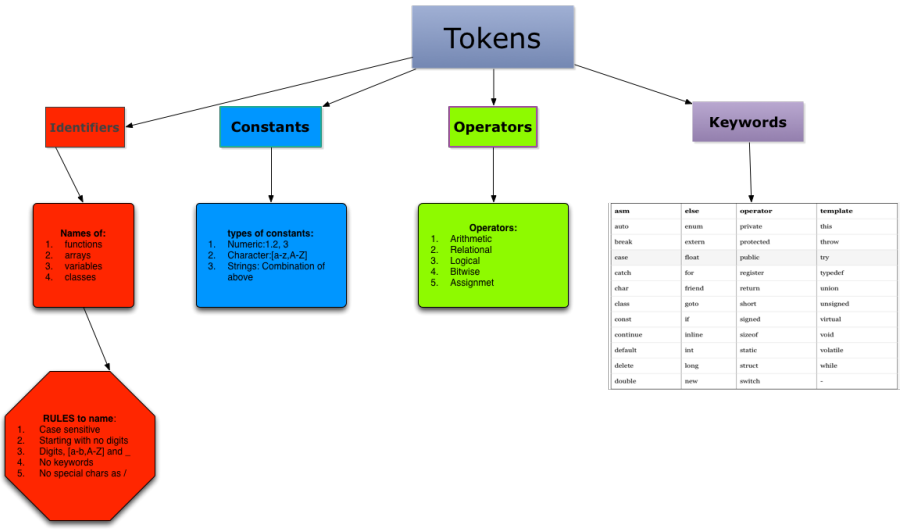
# 1 Tokens in C++

## 1.1 Outline

1. Identifiers.
2. Keywords.
3. Constants.
4. Types.
5. Operators.
6. Rules to name different variables in a programme.
7. Casting.

## 1.2 Tokens




## 1.3 Identifiers

Any name element is an identifier. They are basically names given by the programmer. It is any variable, function, data definition, etc.

### 1.3.1 Rules:

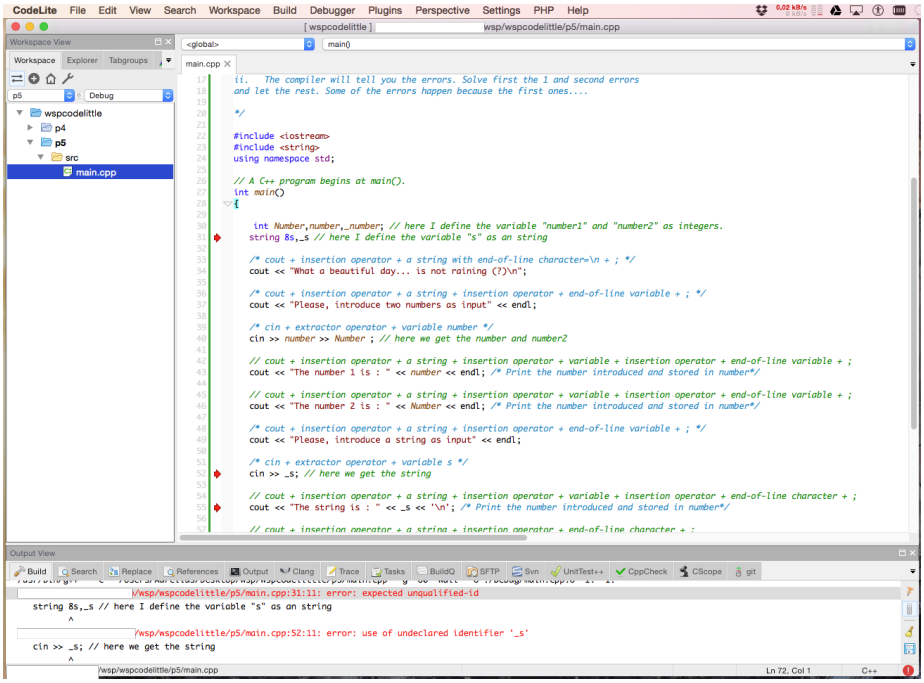
1. Alphabet, digits and underscore are allowed.
2. They can not start with a digit.

3. keywords can be used as identifiers.
4. It is case-sensitive.
5. Special characters are not permitted.

## 1.4 Practice:

1. If you are using an IDE, create a new project.
2. Copy the code from the file identifiers.cpp.
3. Read the program.
4. There are several mistakes, correct them.
5. Compile and run the program.

---




---

Try to answer next questions:

- Is it working?
- Why not?
- What do the compiler say about them?

## 1.5 Keywords and special characters

List of keywords:

| **asm** | **else** | **operator** | **template** |
|---|---|---|---|
| auto | enum | private | this |
| break | extern | protected | throw |
| case | float | public | try |
| catch | for | register | typedef |
| char | friend | return | union |
| class | goto | short | unsigned |
| const | if | signed | virtual |
| continue | inline | sizeof | void |
| default | int | static | volatile |
| delete | long | struct | while |
| double | new | switch | - |

---

List of special characters:

| Special Characters Type | Example |
|---|---|
| Arithmetic Operators | – + * / % |
| Logical Operators | & ! |
| Brackets | ( ) { } [ ] |
| Relational Operators | < > = # |
| Other Symbols | : ; _(underscore) >> ? |

They can not be used as identifiers (named variables, functions, etc. )

## 1.6 Practice

Identify the keywords and special characters, using the previous list, in the program identifiers.cpp.

## 1.7 Constants

- Integer or floating point number, such as 8 or 19.27.
- Single letters between single quotation marks, such as `'\n'` or `'a'`.
- Strings between double quotation marks, such as `"Hello world\n"`.

## 1.8 Fundamental types

You use them to define variables of type:

- `void` (empty type): represent nothing(?).
- `bool` : can be only `true` or `false`;
- `char` : can store characters such as `'a'`, `'b'`, `'\n'`, `'\t'`, etc.
- `int` : can store integer numbers.
  - Modifiers (Signedness): `signed/unsigned`
  - Size: `short/long/long long`

---

## 1.9 Properties of integers

| Type specifier | Equivalent type | Width in bits by data model | | | | |
|---|---|---|---|---|---|---|
| | | C++ standard | LP32 | ILP32 | LLP64 | LP64 |
| `short` | `short int` | at least **16** | **16** | **16** | **16** | **16** |
| `short int` | | | | | | |
| `signed short` | | | | | | |
| `signed short int` | | | | | | |
| `unsigned short` | `unsigned short int` | | | | | |
| `unsigned short int` | | | | | | |
| `int` | `int` | at least **16** | **16** | **32** | **32** | **32** |
| `signed` | | | | | | |
| `signed int` | | | | | | |
| `unsigned` | `unsigned int` | | | | | |
| `unsigned int` | | | | | | |
| `long` | `long int` | at least **32** | **32** | **32** | **32** | **64** |
| `long int` | | | | | | |
| `signed long` | | | | | | |
| `signed long int` | | | | | | |
| `unsigned long` | `unsigned long int` | | | | | |
| `unsigned long int` | | | | | | |
| `long long` | `long long int` (C++11) | at least **64** | **64** | **64** | **64** | **64** |
| `long long int` | | | | | | |
| `signed long long` | | | | | | |
| `signed long long int` | | | | | | |
| `unsigned long long` | `unsigned long long int` (C++11) | | | | | |
| `unsigned long long int` | | | | | | |

---

- Briefly, floating point types represent real numbers:
- `float`: size usually is 16 bits.
- `double`: size usually is 32 bits.
- `long double`: size usually is 64 bits.

---

Range of validity of each type

| Type | Size in bits | Format | Value range | |
|---|---|---|---|---|
| | | | Approximate | Exact |
| character | 8 | signed (one's complement) | -127 to 127 (until C++14) | |
| | | signed (two's complement) | -128 to 127 | |
| | | unsigned | 0 to 255 | |
| integral | 16 | signed (one's complement) | $\pm 3.27 \cdot 10^4$ | -32767 to 32767 |
| | | signed (two's complement) | | -32768 to 32767 |
| | | unsigned | 0 to $6.55 \cdot 10^4$ | 0 to 65535 |
| | 32 | signed (one's complement) | $\pm 2.14 \cdot 10^9$ | -2,147,483,647 to 2,147,483,647 |
| | | signed (two's complement) | | -2,147,483,648 to 2,147,483,647 |
| | | unsigned | 0 to $4.29 \cdot 10^9$ | 0 to 4,294,967,295 |
| | 64 | signed (one's complement) | $\pm 9.22 \cdot 10^{18}$ | -9,223,372,036,854,775,807 to 9,223,372,036,854,775,807 |
| | | signed (two's complement) | | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| | | unsigned | 0 to $1.84 \cdot 10^{19}$ | 0 to 18,446,744,073,709,551,615 |
| floating point | 32 | IEEE-754 | $\pm 3.4 \cdot 10^{\pm 38}$ (~7 digits) | • min subnormal: $\pm 1.401,298,4 \cdot 10^{-47}$<br>• min normal: $\pm 1.175,494,3 \cdot 10^{-38}$<br>• max: $\pm 3.402,823,4 \cdot 10^{38}$ |
| | 64 | IEEE-754 | $\pm 1.7 \cdot 10^{\pm 308}$ (~15 digits) | • min subnormal: $\pm 4.940,656,458,412 \cdot 10^{-324}$<br>• min normal: $\pm 2.225,073,858,507,201,4 \cdot 10^{-308}$<br>• max: $\pm 1.797,693,134,862,315,7 \cdot 10^{308}$ |

## 1.10 Casting

### 1.10.1 i.e., transformation in a different type

You can use a variable of a certain type as it were a variable a different types. For example:

```
int intvar = (int) floatvar
```

where `floatvar` is a variable of type float.

In `C++` it is also possible to use the following syntax:

```
int intvar = int(floatvar)
```

## 1.11 Practice

- Create a new project (if you are using an IDE).
- Copy the code in the file ctetypes.cpp iside inside your new `main.cpp` file.
- Read carefully the program statements and the comments above them.
- Compile the code.
- Execute the code (in an IDE these last two steps often goes together)

---

Answer these questions:

- Is there any mistake?
- Where?
- What does the compiler say?

Then change the variable `s_string` with `i_number`, compile and run...

## 1.12 Assignment Operators

| Operator | Description | Example |
|---|---|---|
| = | assignment operator | A = 10 |
| += | Increase a certain number | A += 2 (A = 6) |
| -= | Decrease a certain number | A -= 3 (A = 1) |

## 1.13 Arithmetic Operator

```
A=4;
A=2;
```

| Operator | Description | Example |
|---|---|---|
| + | Adds two operands or variables | A + B = 6 |
| - | Subtracts second operand from the first | A - B = 2 |
| * | Multiplies both operands | A * B = 8 |
| / | Divides numerator by denominator | A / B = 2 |
| % | After dividing the numerator by denominator remainder will be returned after division | A % B= 0 |
| ++ | Increment operator will increase integer value by one | A++ = 5 |
| -- | Increment operator will decrease integer value by one | A– = 3 |

## 1.14 Relational Operator

```
A=4;
A=2;
```

| Symbol | Meaning | Example |
|---|---|---|
| > | Greater than | A > B returns true |
| < | Less than | A < B returns false |
| >= | Greater than equal to | A >= B returns false |
| <= | Less than equal to | A <= B returns false |
| == | Equal to | A == B returns false |
| != | Not equal to | A != B returns true |

## 1.15 Logical Operators:

```
A=4;
A=2;
```

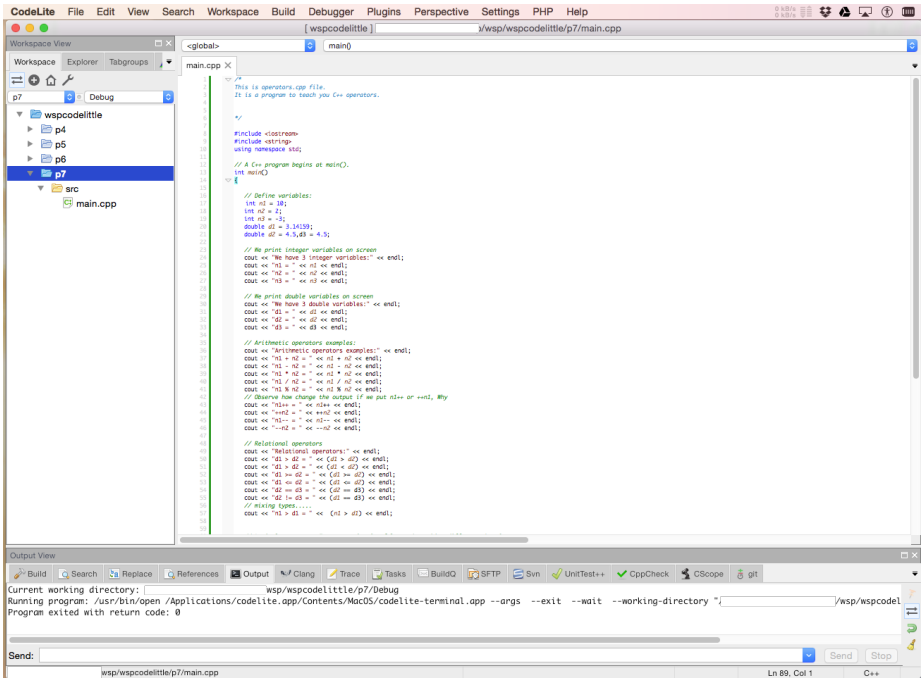
| Operator | Description | Example |
|---|---|---|
| Logical AND (&&)<br>Logical OR (\|\|) | If both operands are non-zero then only condition becomes true If both operands are zero then only condition becomes false | (A && B) is false ( A \|\| B) is true |
| Logical Not (!) | It will reverses the sate of its operand i.e. true will become false | !(A) is true |

## 1.16 Practice:

- Create a new project if you have an IDE. Otherwise, just create a new file.

- Copy the source code of the file operators.cpp on to the new `main.cpp` file.
- Read carefully the program statements and the comments above them.
- Compile/run the code.

---




# 2 Statements and decision making

## 2.1 Outline

- Define the concept of program statement.
- Define the concept of program flow.
- Implement a decision-making program:
  - `if` structure.
  - `if-else` structure.
  - nested `if-else` structure.
  - `switch` statement.
  - `switch` is equivalent to nested `if-else`.

## 2.2 Statements

C++ statements are individual instructions of a program:

- They finish with a semicolon, i.e, `;` .
- They are executed in the same order as they appear in the source code.

We have already seen several statements, e.g.,

- Variable declarations.

  ```
  int x;
  ```

- Print on the stardard output `std::cout`.

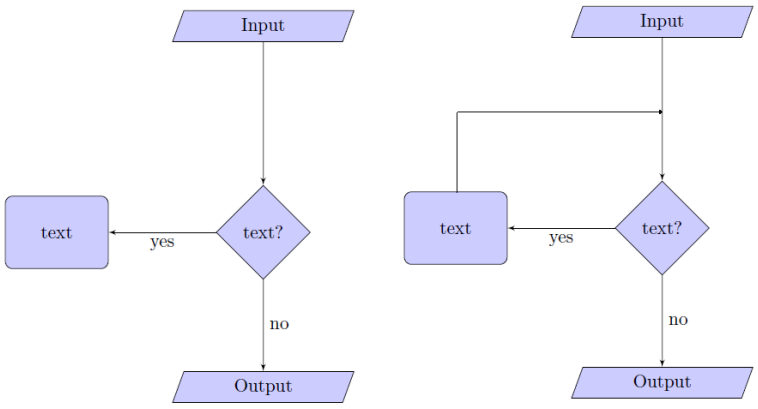
  ```
  cout << "x is 100";
  ```

- Read from stardard intput `std::cin`.

  ```
  cin >> x ;
  ```

## 2.3 Program Flow and control flow statements.

A program can be defined as the ordered execution of several computer statements. The execution order is from the top to the bottom. The control flow statements allows the program to prevent (Decision making) or repeat (Loop statements) the execution of some statements. Here you can see two program flow diagrams of a very silly program:



## 2.4 Decision Making

- `if ( condition ) statement;`

  If the condition is false, then the statement is not executed, e.g.,

```
if (m == 18)
cout << "m is = 18";
```

- `if ( condition ) { statement 1; statement 2;}`

  When you need several statements under the same condition, embrace them with curly brackets, i.e.,

```
if (m == 18) {
 cout << "m is ";
 cout << m;
}
```

The statements within the curly brackets form a code *block*. It is strongly suggest to use the braces, even with only one statement.

- `if ( condition ) statement1; else statement2;` Execute statement1 if the condition is true, otherwise execute statement2. You can replace one of the two statements with a block, for example:

```
if (m == 18) {
 cout << "m is 18";
 cout << "You are an adult person";
}
else
cout << "m is not 18";
```

- Nested `if` conditions. No need to combine several if-else constructs as it is possible to use the construct

```
if (m > 0)
 cout << "m is positive";
else if (m < 0)
 cout << "m is negative";
else
 cout << "m is 0";
```

- `switch`. Similar to the nested if-else, sometimes is faster. It can be easily used when the control variable has a constant value. The syntax is

```
switch (expression) {
 case constant1:
   block1;
   break;
 case constant2:
   block2;
   break;
 default:
   default-block;
}
```

For example:

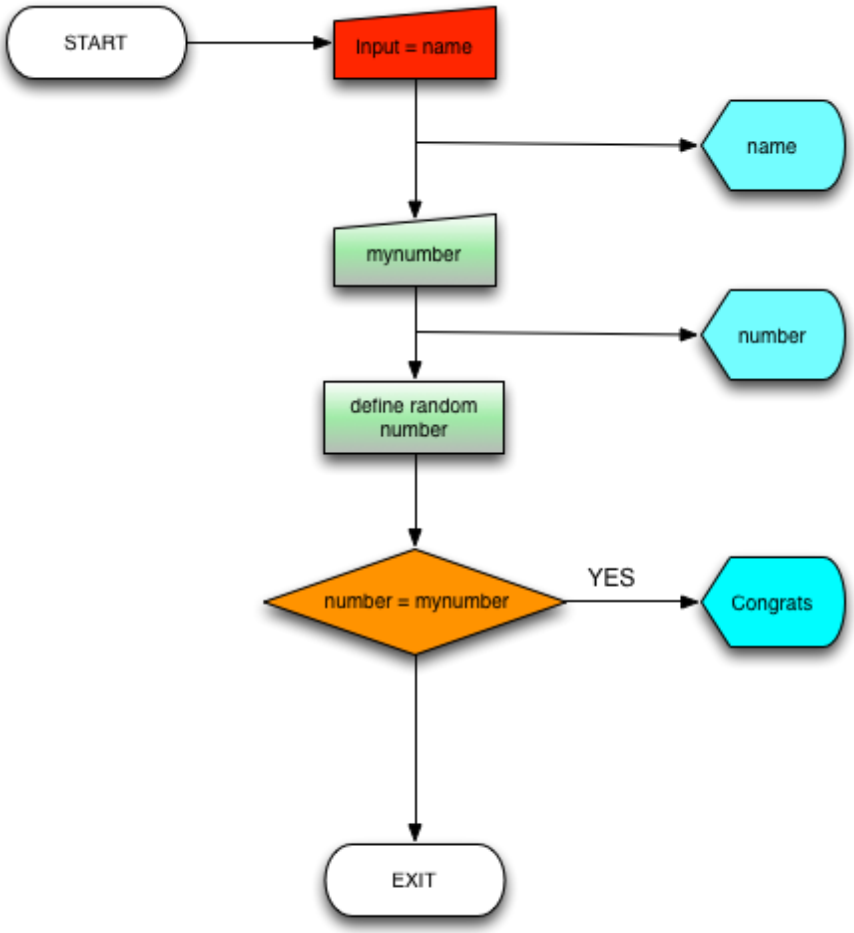
```
switch (x) {
  case 1:
  case 2:
  case 3:
     cout << "x is 1, 2 or 3";
     break;
  default:
     cout << "x is not 1, 2 nor 3";
}
```

## 2.5 Practice

- Create a new project in the workspace.
- Copy the source code of the file decision_making.cpp to the new main.cpp file:
- Read carefully the statements of the program before playing with it!
- Try to figure out what is the output.
- Don't you think that something is missing?

---

Here there is the flow diagram that might be help you to understand this program

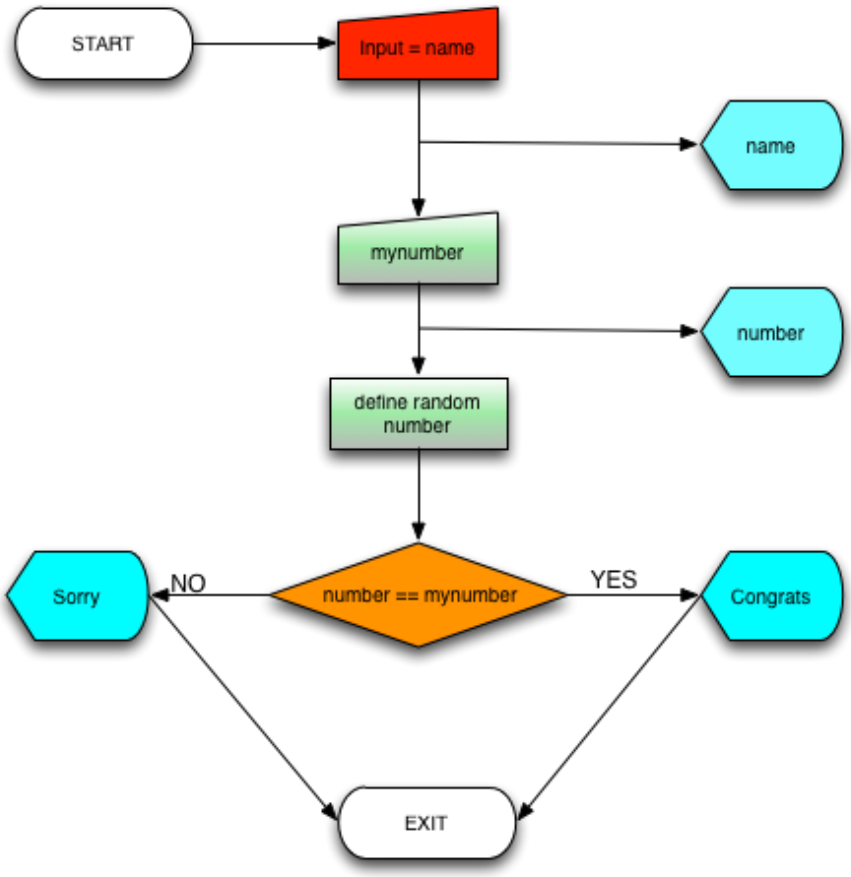

START
Input = name
name
mynumber
number
define random number
number = mynumber
YES
Congrats
EXIT

---

- Now create another project and deploy the file decision_making2.cpp
- What should be the code in case you want to give a second opportunity to the programme user?

---

Here there is the flow diagram that might be help you to understand this program



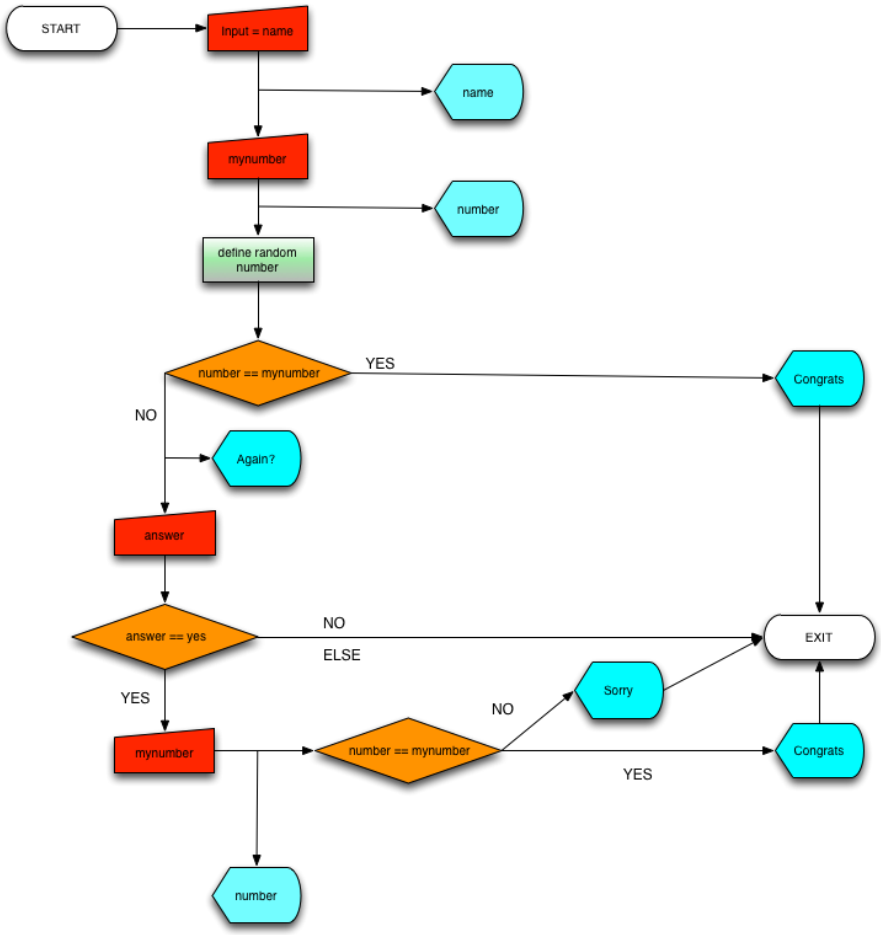

---

Let's do something more interesting. * Create one project more and deploy the file (decision_making3.cpp)[../codes/decision_making3.cpp] * Read the code! *

What happens if the input in line 32 is not a text but a number? * How will you change it in order to prevent such case?

---

Here there is the flow diagram that might be help you to understand the



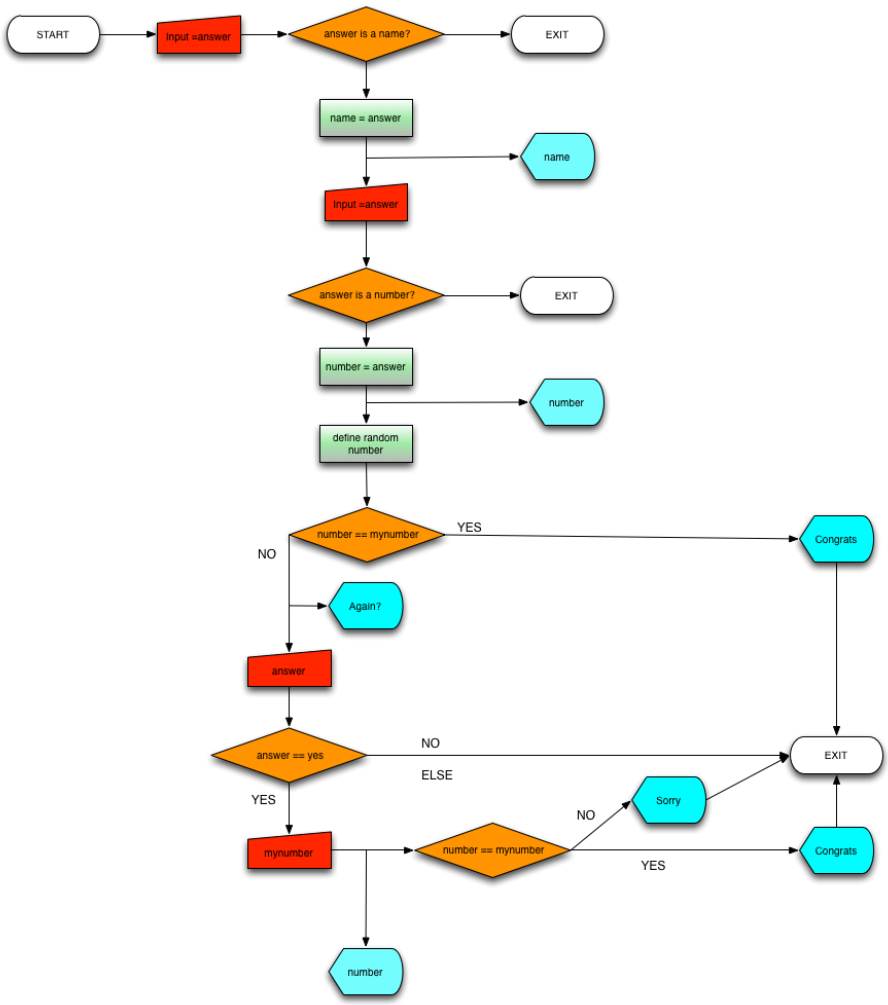

---

- Another exercise... create a new project based on decision_making4.cpp.
- Try to figure out how this program work!
- Can you be even more specific? for example, in line 77 the answer can be not a number... How can you change the code in order to prevent this case?

- Can you find or think other cases where the programme might not work?

---

The flow diagram is




---

Last but not least...

- create a new project based on decision_making5.cpp

- Read carefully the statements of the programme.
- Try to understand what is the possible output as a function of the possible input.
- Switch-statement can be easily transformed in a nested if-else statement. Can you please do it?

---

The flow diagram is