```
# include <iostream>
```
→ library

```
int  main ( )
{
        std ::cout  <<  " Hi world!" ;
        return  0;
}
```

---

int width;

width = 5 ; → giving the variable a value (assignment)

width = 7 ;

int width = 5 ; copy initialization

int width (5) ; direct initialization

int width {5} ; uniform initialization in C++11

↳ allows for zero initialization { }

↳ will lead to an error if given non-integer value (eg 4.5)

↳ copy and direct will drop the fraction to give 4.

```
int a, b ;
   ⋮
a = 5 ;
b = 6 ;
```
} neater to split?

```
int  a = 5 , b = 6 ;
int  c (7) , d (8) ;
int  e {9} , f {10} ;
```
} multiple initialization

✗ int a, b = 5

```
# include  <iostream>
```
→ input/output stream library

```
std :: cout  <<  " Hello  World!" ;
```

$\underbrace{\text{std}}_{\text{standard}}$ :: $\underbrace{\text{cout}}_{\text{character output}}$

↳ can also take / print numbers, and value of variables

```
# include <iostream>

int main()
{
   std::cout << "Hello!";
   std::cout << "My name is Iman";
   return 0;
}
```

=> Hello! My name is Iman (Printed on the same line)

...

```
{
   std::cout << "Hello" << std::endl;
   std::cout << "My name is Iman";
}
```

→ will cause the cursor to move to the next line of the console

=> Hello
My name is Iman

- moves cursor and flushes the output

...

```
{
   int x{5};
   std::cout << "x is equal to:" << x << '\n';
   std::cout << "And that's all folks!\n";
   return 0;
}
```

→ using \n standalone

→ using \n in a double quoted piece of text

=> x is equal to 5
And that's all folks!

(strings can also be initialized)
string mystring { };

...

```
# include <iostream>

int main
{
  std::cout << "Enter a number:"
  int x{ };
  std::cin >> x;
  std::cout << "You entered" << x << '\n';
  return 0;
}
```

int x{ }; → define a variable to store user input (zero initialize)

std::cin >> x; → input

uninitialized => `int x ;`

(random value)

identifier => name of a variable, type etc.

↳ cannot be keyword / start with a number

* Always initialize your variables

* undefined behaviour => most likely a result of unitialized variable
(program works anyways but it is wrong)

* C++ is case sensitive

→ variables => normally one word, all lower case

→ identifier names starting with a capital letter are normally used for user-defined types

- Literals are fixed values that have been inserted directly into the source code

  ↳ literal constant (can't be changed)

- Operators, operands

+, -, x, / | literals (eg. 2, 3)

=, ==, <<, >> → PEMDAS

- Unary, Binary, Ternary

↓ work on one operand eg. -5

↓ works on 2 operands 2+3

↓ works on 3 operands

---

expressions don't compile by themselves, requires a statement

```
int x {2+3};
```

type identifier {expression}

```
2+3;
```

(useless, value will be discarded)

Functions

```
                  → function name
void  identifier ( )
{                         function body
// code here
}
```

```
Starting  main ( )
In  doPrint ( )
Ending  main ( )
```

```
# include  <iostream>

void  doPrint ( )
{
  //
}

int  main
{                                                        function defined  beforehand
  doPrint ( );  → main is the caller of the
                  ( function  call )
   :
}
```

* Functions are able to call other functions as well

Function return values

- return type (type defined before the function's name)
  ↳ void, int...
- return statement indicates return value
- if return statement is void, std::cout << returnNothing(); will give an error.
- Always provide a return value for any function that has a non-void return type
- Failure to return a value from a function with a non-void return type (other than main) will result in undefined behaviour
-

Data Types

- signed / unsigned (` - ´)
- character (a, b, $ ...)
- numerical (1, 2, ...)
- Boolean (true, false)
- floating point (3.14, 0.01)

string has its own library to be included

```
#include <string>
    string mystring;   ← initialize the string
     mystring = "This is a string";
```

75

0113 ??

0x4b ??

base 8

base 16

$$75_{10} = 7 \times 10^1 + 5 \times 10^0$$

$$= 75$$

$$1 \times 8^2 + 1 \times 8 + 3 = 64 + 8 + 3$$

$$= 75$$

$$16 \times 4 = 64 + 8$$

$$= 7$$

&&

a	b	a&&b
T	T	T
T	F	F
F	T	F
F	F	F

or

a	b	a‖b
T	T	T
T	F	T
F	T	T
F	F	F

$7 == 5 \;?\; \underset{\text{True}}{a} : \underset{\text{False}}{\underline{b}}$

↳ if the statement is

if else

if true

```
if      ( x==100)
    cout    << "x is 100" ;
```

while loops

```
{ int  n=10;
  while   (n>0) {
       cout  <<  n <<  ", " ;
       --n;
  }
  cout  <<"liftoff!\n";
}
```

=> 10, 9, 8, 7, ..., 1, 0, liftoff

do loop

```
{
  do  {
       :
      } while  (str != "goodbye");
}
```

for loop

```
{
 for   (int  n=10; n>0; n--) {
       cout  << n << ", ";
 }
 cout << "liftoff ! \n";
}
```

Range based loops

```
int  main ()
{
   string  str {"Hello"};
   for  (char : str)
   {
     cout <<  "[" <<c<< "]";
   }
   cout << '\n';
}
```

```
if ( n==3)
  { break }
```

break; => stops loop

continue; => causes the loop to skip the rest of the steps in the current iteration

goto => allows to jump to another point in the program

```
{
  int   n=10;
mylabel:
   cout << n << ", ";
   n--;
   if  (n>0 )  goto  mylabel;
   cout  << "liftoff ! \n";
}
```

switch

x

```
switch (expression)
{
    case   constant 1 :                          x==1
           group-of-statements-1;
           break;
    case   constant 2 :                          x==2
           group-of statements -2;
           break ;
     :
   default :
        default-group-of_statements
```

```
case 1 :
case 2 :   } x is 1,2 or 3
case 3 :
 - - - - -
```

void duplicate

- references are indicated with an ampersand (&)

```
void duplicate (int& a , int& b , int& c)
                     ↕        ↕        ↕
{      duplicate (   x        y        z  )
    a* 2
    b* 2
    c* 2
  }
```

```
string concatenate (string a , string b)
{
   return a+b
 }
```