

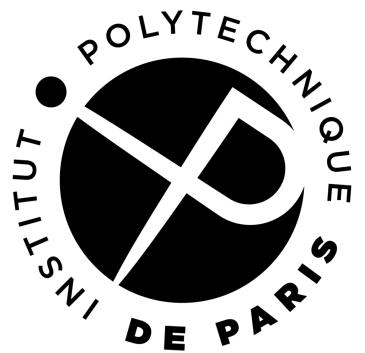
Operators and algorithmic structures

Bachelor of Science - École polytechnique

gael.thomas@inria.fr

Key concepts

- Operators
- Algorithmic structure: `if`, `switch`, `while`, `for`, `do while`



I. Operators

Operators (1/2)

- Arithmetic: +, -, *, /, % (modulo)
- Comparison: <, <=, >, >=, == (equal), != (not equal)
 - $x == 42 \Rightarrow$ false (0) if x not equal to 42, true (a positive number) otherwise
- Logical: && (and), || (or), ! (not)
 - For example $(x == 1) \&\& !(y == 2)$
- Assignment: =, +=, -=, *=, /=, %=
 - $a = 42 \Rightarrow$ give the value 42 to a
 - $b += 3 \Rightarrow$ add 3 to b

Operators (2/2)

- Unary: `++` (increment), `--` (decrement)
 - `1 + (x++)` => compute $(1 + x)$ and then increment `x`
 - `1 + (++x)` => increment `x` and then compute $(1 + x)$
- Bitwise: `&` (and), `|` (or), `^` (xor), `<<` (shift left), `>>` (shift right), `~` (invert all the bits)

Note the difference between a logical and a bitwise operator:

```
(1 && 2) != (1 & 2)
```

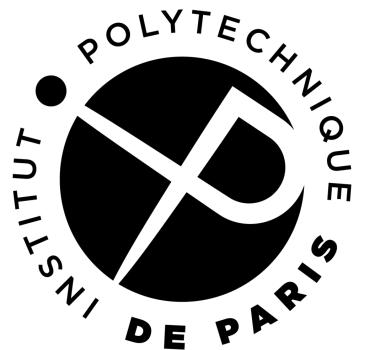
* `1` (true) `&&` `2` (true) = true = a value not equal `0`

* `1 & 2 = 0` because

`01`

`& 10`

`00`



II. Algorithmic structures

Algorithmic structures and blocks

- The structure of a C program is given by the blocks
 - Starts with a { and ends with a }
 - A block groups a set of statements together
- Contrary to python, indentation has no meaning
 - But we usually indent the code contained in a block

```
int main(int argc, char** argv) {  
    if(0 == 1) {  
        printf("This computer is strange\n");  
        printf("It thinks that 0 is equal to 1\n");  
    }  
    return 0;  
}
```

} Block
executed if 1
is equal to 0

Conditional

```
if(cond)
    statement
else
    statement
```

```
if(x == 42) {
    printf("x is equal to 42\n");
} else if(x == 666) {
    printf("x is equal to 666\n");
} else {
    printf("x is different\n");
}
```

Note: formally, in this case, we can omit the braces since we have a single statement. However, we advise you to systematically use braces even if it's not required to avoid mistakes

```
if(x == 42)
    printf("x is equal to 42\n");
    printf("Bad identitation is misleading here!\n");
```

Ternary operator

- The ternary operator is used to build a short conditional

```
x = cond ? expr1 : expr2
```

=> x takes the value expr1 if cond is true, expr2 otherwise

```
res = x < 42 ? 0 : 666;
```

Switch

- var has to be an integer variable (char, short, int...)
(does not work if var has another type)

```
switch(var) {  
    case v0: ... break;  
    case v1: ... break;  
    case v2: ... break;  
    ...  
    default: ...  
}
```

```
int x = 42;  
  
switch(x) {  
    case 0: printf("0\n"); break;  
    case 1: printf("1\n"); break;  
    default: printf("Other\n");  
}
```

Note: if you omit a break, the execution continue with the next case

While loop

while(cond)
statement

```
int tab[10];
int i = 0;

while(i < 10) {
    tab[i] = i*2;
    i++;
}
```

For loop

- A for loop is a shortcut for a while loop

```
for(init; cond; iter)  
    statement
```



```
init;  
  
while(cond) {  
    statement  
    iter;  
}
```

```
int tab[10];  
  
for(int i = 0; i < 10; i++) {  
    tab[i] = i*2;  
}
```



```
int tab[10];  
int i = 0;  
  
while(i < 10) {  
    tab[i] = i*2;  
    i++;
```

Do while loop

- Useful when you want to test the condition at the end

while(cond)
statement

```
int val;  
  
do {  
    val = rand();  
} while(val != 3)
```

rand() is a function that returns a random number

Congratulation!

You now understand 80% of the C language!

Key concepts

- Operators
- Algorithmic structure: `if`, `switch`, `while`, `for`, `do while`