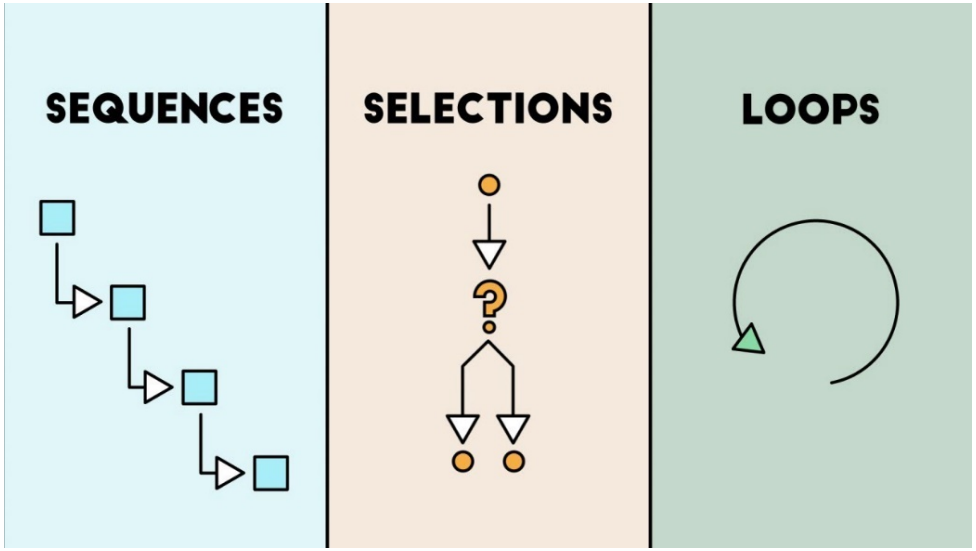
Control Structures (Part 1)

Control Structures

What is Control?

- Which piece of code to be executed next?



Three Basic types:

- Sequence
- Conditional – If (…) do this
- Iteration – Repeat this until (…)

Outline

- **Part 1: Boolean data type**
- Part 2: Relational and Logical Operators
- Part 3: Selection Control Structure (`if-else`)

Boolean Type

- In general, a *Boolean* data type takes two possible values: either *true* and *false*.
- But C language does not have a Boolean data type;
- C treats **zero** as false and **non-zero** as true.
  - Later on, C++ has the "bool" data type
- A Boolean expression in C language evaluates to either 1 (true) or 0 (false).

Outline

- Part 1: Boolean data type
- **Part 2: Relational and Logical Operators**
- Part 3: Selection Control Structure (`if-else`)

2. Relational Operators

Operators	Examples	Meanings
Equality		
==	x == y	x is equal to y?
!=	x != 10	x is not equal to 10?
Relational		
>	x > 3	x is greater than 3?
<	y < 0	y is less than 0?
>=	x >= y	x is greater than or equal to y?
<=	y <= -4	y is less than or equal to -4?

2.1. Results of Comparison

- The result of a comparison is either `1` (true) or `0` (false).
- Suppose **x** and **y** are declared as

```
int x = 0 , y = 100 ;
```

```
x > 10                  ➔ 0
y >= ( x + 100 )        ➔ 1
100 == ( x + y )        ➔ 1
2.4 / 24 == 0.1         ➔ ??
```

Note: since floating point numbers **may not be exact**, be careful when checking the equality of floating points numbers, e.g., check if |a – b| is very small.
(see some examples soon)

An Example

- What is the output of the following program?

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int x = 0   ;
6      int y = 100 ;
7
8      printf( "%d\n" , x >  10       );
9      printf( "%d\n" , y >= (x+100) );
10     printf( "%d\n" , 100 == (x+y) );
11
12     return 0 ;
13 }
```

```
0
1
1
```

One more example…

- What is the program output?

Note: since floating point numbers **may not be exact**, be careful when checking the equality of floating points numbers, e.g., check if |a – b| is very small.

```
??
??
??
```

```
1  #include <stdio.h>
2
3  int main()
4  {
5    double a = 1.0/2.0 ;
6    double b = 1.0/5.0 ;
7    double c = 1.0/7.0 ;
8    printf( "%d\n" , a + a                       == 1.0 );
9    printf( "%d\n" , b + b + b + b + b           == 1.0 );
10   printf( "%d\n" , c + c + c + c + c + c + c == 1.0 );
11
12   return 0 ;
13 }
```

One more example…

- What is the program output?

```
1  #include <stdio.h>
2
3  int main()
4  {
5    double a = 1.0/2.0 ;
6    double b = 1.0/5.0 ;
7    double c = 1.0/7.0 ;
8    printf( "%d\n" , fabs( a + a - 1.0 )                          < 1e-7 );
9    printf( "%d\n" , fabs( b + b + b + b + b - 1.0 )              < 1e-7 );
10   printf( "%d\n" , fabs( c + c + c + c + c + c + c - 1.0 ) < 1e-7 );
11
12   return 0 ;
13 }
```

Note: since floating point numbers **may not be exact**, be careful when checking the equality of floating points numbers, e.g., check if |a – b| is very small.

```
1
1
1
```

1e-7 means “one” times “10 to the power -7”, i.e., 0.0000001

2.2. Logical Operators

! (Not) **&&** (And) **||** (Or)

- Operator **!** is unary, taking only one operand.
- Operators **&&** and **||** are binary, taking two operands.
- All logical operators yield either 1 (true) or 0 (false).

2.2.1. Evaluating Logical AND operator (&&)

a	b	a && b
0	0	0
0	1	0
1	0	0
1	1	1

This is called the **Truth Table**, which is useful for logical operations

- The result is true only when both operands are true.

e.g., to express "x is positive and x is less than 10",

(x > 0) && (x < 10)

e.g., eng_score > 50 && chi_score > 50

2.2.2. Evaluating Logical OR operator (||)

a	b	a || b
0	0	0
0	1	1
1	0	1
1	1	1

This is called the **Truth Table**, which is useful for logical operations

- The result is true if either operand is true.

e.g., to express "x is equal to 2 or 3", one can write

(x == 2) || (x == 3)

e.g., eng_score > 50 || chi_score > 50

2.2.3. Evaluating Logical NOT operator (!)

a	!a
0	1
1	0

e.g., to express "x is not a positive number",
one can write

```
!( x > 0 )
```

e.g., ! (score == 100) is the same as score != 100

2.3. Operand to Logical Expressions

- When a value is treated as a Boolean value
  - A zero is treated as false.
  - Any non-zero value is treated as true.

- So, we may actually write:

  5 && 5 ➔ 1 (5 is treated as true)

  !5 ➔ 0

  !!5 ➔ !0 ➔ 1

Note: avoid writing code like this… think about **readability**
Don't write code like this!!!

2.4. Operator precedence and associativity

Operator	Associativity	Precedence
( ) ++ (postfix) -- (postfix)	left to right	Highest
+(unary) -(unary) ++(prefix) --(prefix) !	right to left	
* / %	left to right	
+ -	left to right	
< <= > >=	left to right	
== !=	left to right	
&&	left to right	
||	left to right	
?:	right to left	
= += -= *= /= etc.	right to left	
, (comma operator)	left to right	Lowest

Exercise: Evaluating Boolean Expressions

- Suppose **x** is 0, **y** is 10. Evaluate the following expressions:

a) `x < 5 || !( y < 1 )`

b) `x + y > -1 && y % 2 == 0`

c) `!y`

d) `(x >= 0) + (y >= 0)`

e) `5 < x < 10`

This one is really tricky!!!
In fact, inappropriate in C,
but Python allows…

Continue from last example on prev. page

- The following program example illustrates why the expression **5<x<10** strangely gives a result of 1:

```
1   #include <stdio.h>
2
3   int main()
4   {
5       int x = 0 ;
6
7       printf( "%d\n" ,  5 < x        );
8       printf( "%d\n" , (5 < x) < 10 );
9       printf( "%d\n" ,       0 < 10 );
10      printf( "%d\n" ,  5 < x  < 10 );
11      return 0;
12  }
```

The expression **5<x<10** is understood completely differently by the C compiler. **In C, never use comparison in such form!**

What should be the right expression to use in C?

```
0
1
1
1
```

Example: Expressing Conditions

Note: In this exercise. x and y are variables of type `int`.

- x is a number between 5 and 10 (exclusive):
  - In other words, x is greater than 5 and x is less than 10

```
x > 5 && x < 10
```

- Both x and y are non-zero integers and their sum is an odd number:
  - In other words, x is not equal to 0, and y is not equal to 0, and (x+y) is an odd number.

```
(x != 0) && (y != 0) && ((x+y) % 2 != 0)
```

Exercise: Expressing Conditions

Note: In this examples, x, y, and z are variables of type `int`.

a) Exactly one of x and y is zero
   - In other words, when x is zero, y is not zero, or when y is zero, x is not zero.

b) x, y, and z are identical.

What are the testing cases?

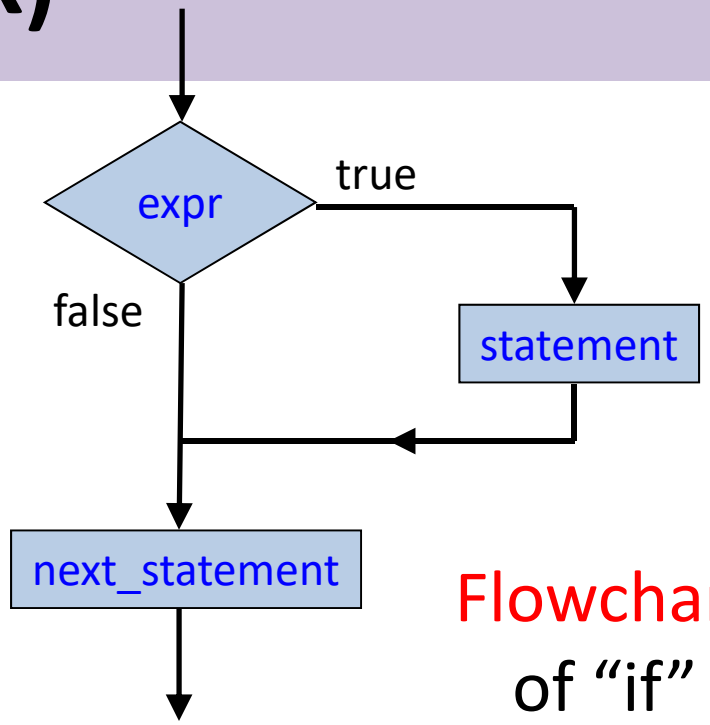
c) Among x, y, and z, x has the largest value and z has the smallest value.

Outline

- Part 1: Boolean data type
- Part 2: Relational and Logical Operators
- **Part 3: Selection Control Structure (`if-else`)**

3. `if` Statement (syntax)

```
if ( expr )
    statement ;
next_statement ;
```



Flowchart of "if"

- Allows us to conditionally perform a task
  - If **`expr`** is true (non-zero), then **`statement`** is executed.
  - Otherwise, computer skips **`statement`**, and control is passed to **`next_statement`**.

Note: Flowchart – allows you to visualize the control flow

3.1. if Statement (Example)

```
1  int score ;
2  printf( "Please enter your score: " );
3  scanf( "%d" , &score );
4
5  if ( score >= 60 )
6      printf( "Passed!\n" );
7
8  if ( score < 60 )
9      printf( "Failed!\n" );
10
11 printf( "Your score is %d.\n" , score );
```

```
Please enter your score: 80↵
Passed!
Your score is 80.
```

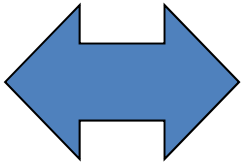
```
Please enter your score: 40↵
Failed!
Your score is 40.
```

3.2. Indentation

What's the output if the condition **`(x > 0)`** is true/false?

```
1  if ( x > 0 )
2      printf( "A" );
3      printf( "B" );
4  printf( "C" );
```

Same as



```
if ( x > 0 )
    printf( "A" );
printf( "B" );
printf( "C" );
```

Indentation – leading white space at the beginning of a line for align code.

Indenting codes does not affect a C program; BUT it helps to make the code easier to read.

- How do we conditionally execute multiple statements then?

3.3. Compound Statement

```
1  if ( score >= 60 )
2  {
3      printf( "You have passed!\n" );
4      printf( "Congratulations!\n" );
5  }
6
7  printf( "Your score is %d\n" , score );
```

Execute all the statements between { and } if score ≥ 60.

- { … } groups multiple statements into ONE *compound statement.*
- No need to have a semicolon (;) after { … }

3.4. Two Common Mistakes

1. Using = instead of == as equality operator

```
if ( a = 0 )
   some_statement ;
```

   – Variable **a** is assigned 0 and the whole expression is evaluated to 0, and 0 means false.

2. Placing ';' after the condition of an `if` statement

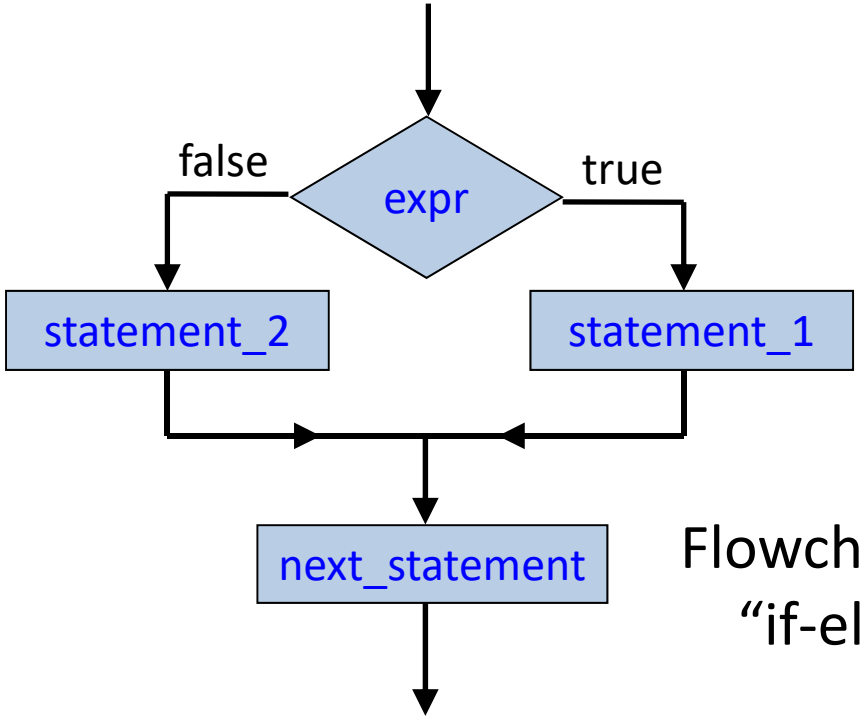
```
if ( a != 0 );
   some_statement ;
```

interpreted as

```
if ( a != 0 ) {
}
some_statement ;
```

3.5. `if-else` Statement (syntax)

```
if ( expr )
    statement_1 ;
else
    statement_2 ;
next_statement ;
```



Flowchart of "if-else"

- Allows us to conditionally perform one of the two tasks
- Why it helps?

See again our previous example…

```
1   int score ;
2   printf( "Please enter your score: " );
3   scanf( "%d" , &score );
4
5   if ( score >= 60 )
6       printf( "Passed!\n" );
7
8   if ( score < 60 )
9       printf( "Failed!\n" );
10
11  printf( "Your score is %d.\n" , score );
```

What is its flowchart?

Can we simplify the code?

```
Please enter your score: 80↵
Passed!
Your score is 80.
```

```
Please enter your score: 40↵
Failed!
Your score is 40.
```

3.5.1. `if-else` Statement (Example)

```
1  int score ;
2  printf( "Please enter your score: " );
3  scanf( "%d" , &score );
4
5  if ( score >= 60 )
6      printf( "Passed!\n" );
7  else                          // Replaces "if (score < 60)"
8      printf( "Failed!\n" );
9
10 printf( "Your score is %d.\n" , score );
```

- If "score >= 60" is true, then "score < 60" must be false, and vice versa. Note: the two conditions are mutually exclusive!!!

3.6. How does `else` pair with `if`?

IMPORTANT RULE: An **`else`** statement attaches to the **nearest `if`** that has not been paired with an **`else`**.

```
if ( x % 2 == 1 )
    if ( x > 100 )
        printf( "A" );
else
    printf( "B" );
```

```
if ( x % 2 == 1 )
  if ( x > 100 )
      printf( "A" );
  else
      printf( "B" );
```

```
if ( x % 2 == 1 )
{
  if ( x > 100 )
      printf( "A" );
  else
      printf( "B" );
}
```

```
if ( x % 2 == 1 )
{
    if ( x > 100 )
        printf( "A" );
}
else
    printf( "B" );
```

Observe the brackets {…}!

3.7. Conditionally performing 1 of N tasks

- An **`if-else`** statement only branches two ways. How do we branch multiple ways?
  - Example: Ask the user to choose among three choices and perform one of three tasks accordingly?

- Solution: Use multiple **`if-else`** statements

3.7. Conditionally performing 1 of N tasks (Version 1)

```
int choice ;
printf( "Please enter your choice (1-3): " );
scanf( "%d" , & choice );

// Carry out the corresponding task
if ( choice == 1 )
    // Carry out task #1

if ( choice == 2 )
    // Carry out task #2

if ( choice == 3 )
    // Carry out task #3
```

Observation: In this example, the conditions are exclusive.

If **`choice`** is 1, there is no need to check if **`choice`** is 2 or 3.

3.7. Conditionally performing 1 of N tasks (Version 2a)

```
int choice ;
printf( "Please enter your choice (1-3): " );
scanf( "%d" , & choice );

// Carry out the corresponding task
if ( choice == 1 )
    // Carry out task #1
else
{
    if ( choice == 2 )
        // Carry out task #2
    else
    {
        if ( choice == 3 )
            // Carry out task #3
    }
}
```

Also known as *nested if-else* statements (i.e., if-else statements within if-else statements)

3.7. Conditionally performing 1 of N tasks (Version 2b)

```
int choice ;
printf( "Please enter your choice (1-3): " );
scanf( "%d" , & choice );

// Carry out the corresponding task
if ( choice == 1 )
    // Carry out task #1
else
    if ( choice == 2 )
        // Carry out task #2
    else
        if ( choice == 3 )
            // Carry out task #3
```

An `if` statement or an `if-else` statement is treated as one statement. Therefore, the { … } surrounding the inner `if-else` statements are optional.

3.7. Conditionally performing 1 of N tasks (Version 2c)

```
int choice ;
printf( "Please enter your choice (1-3): " );
scanf( "%d" , & choice );

// Carry out the corresponding task
if ( choice == 1 )
    // Carry out task #1
else
if ( choice == 2 )
    // Carry out task #2
else
if ( choice == 3 )
    // Carry out task #3
```

Different style of indentation

3.7. Conditionally performing 1 of N tasks (Version 2d)

```
int choice ;
printf( "Please enter your choice (1-3): " );
scanf( "%d" , & choice );

// Carry out the corresponding task
if ( choice == 1 )
    // Carry out task #1
else
if ( choice == 2 )
    // Carry out task #2
else
    // Carry out task #3
```

We may omit the last test: exclusive cases

3.7. Conditionally performing 1 of N tasks (Version 2d)

```
int choice ;
printf( "Please enter your choice (1-3): " );
scanf( "%d" , & choice );

// Carry out the corresponding task
if ( choice == 1 ) {
    // Carry out task #1
}
else
if ( choice == 2 ) {
    // Carry out task #2
}
else {
    // Carry out task #3
}
```

and we may have { … } for compound statements

3.7. Conditionally performing 1 of N tasks (Version 2e)

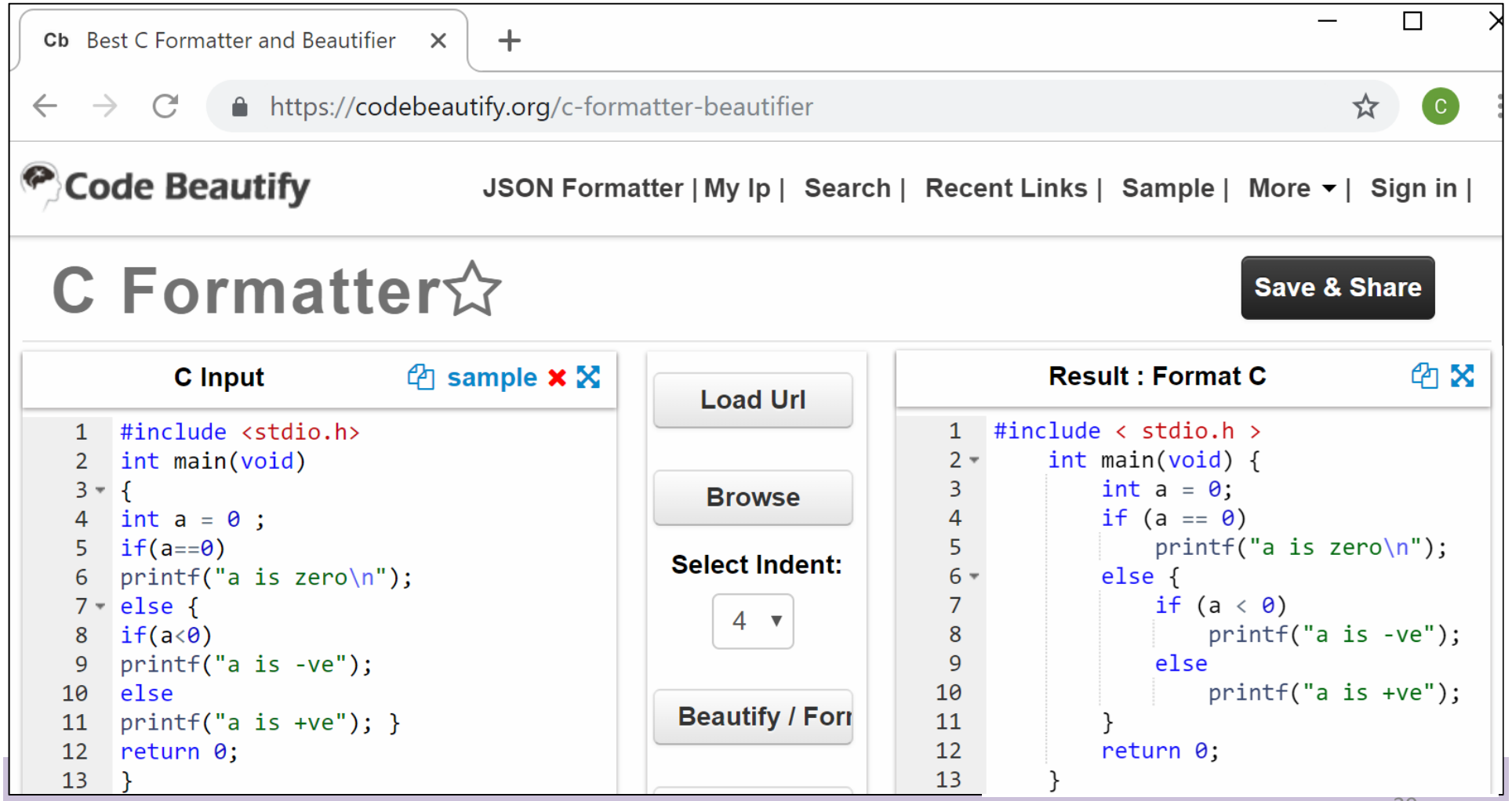
```
int choice ;
printf( "Please enter your choice (1-3): " );
scanf( "%d" , & choice );

// Carry out the corresponding task
switch ( choice )
{
    case 1 :
      // Carry out task #1
      break ;
    case 2 :
      // Carry out task #2
      break ;
    default : // if not cases 1 and 2, so case 3
      ...
}
```

In C language, you may also use “switch”

Code Beautifier: code is also for human to read

- Indentation helps to enhance code readability



https://codebeautify.org/c-formatter-beautifier

Summary

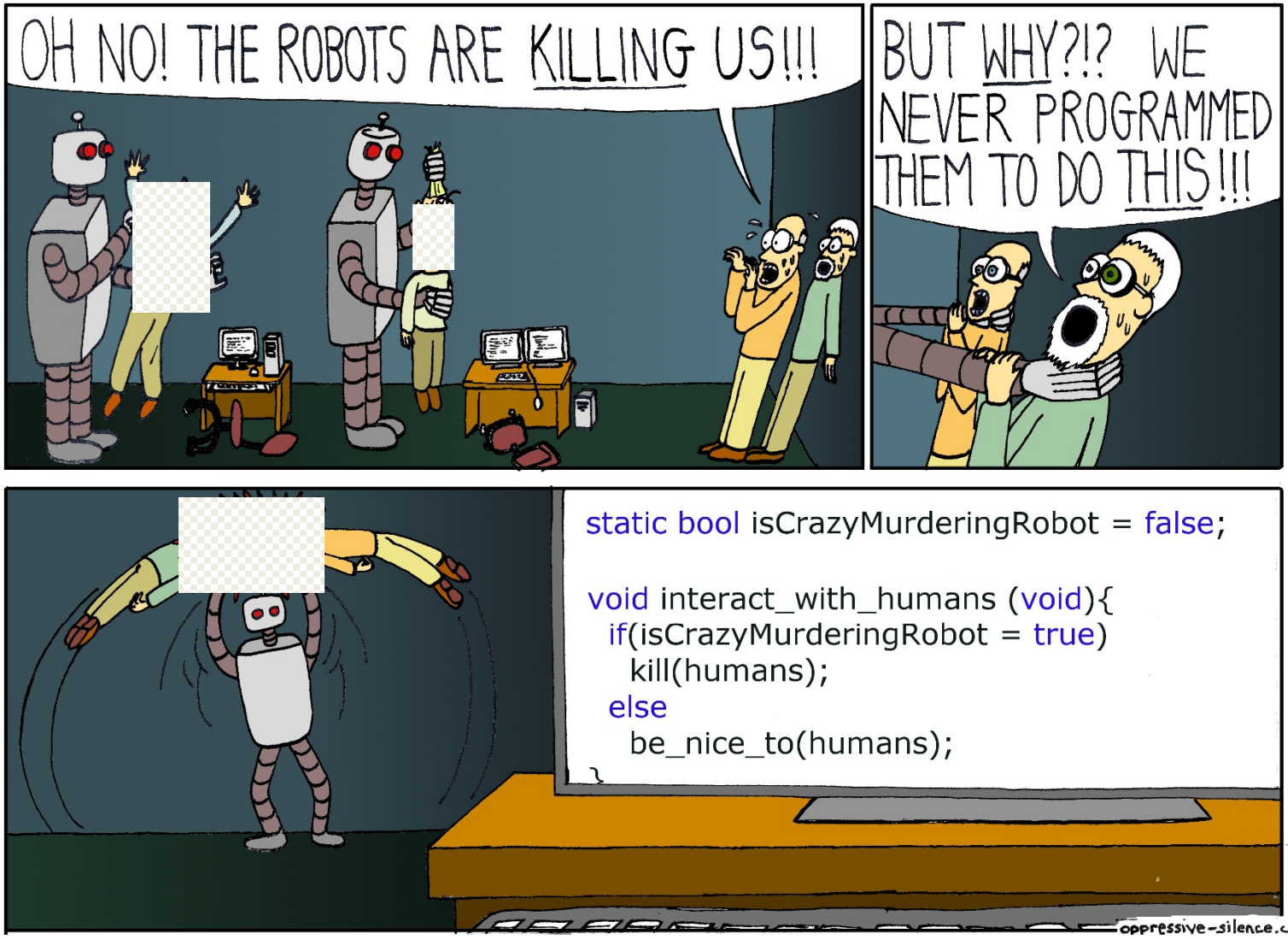
- Relational Operators and Logical Operators

  ==, !=, <, >, <=, >=, !, &&, ||

- if-else statement

- Nested if-else statement

“NOT” really for fun…



A Common Mistake I’ve Mentioned earlier

From https://www.reddit.com/r/ProgrammerHumor/comments/g0o9mb/oh_no/