`if` Lesson 1 Outline

Absolute Value

Consider the function

$$a(y) = |\, y \,|$$

So we know that

$$\begin{aligned}
a(-2.5) &= |\ -2.5\ | = +2.5 \\
a(-2) &= |\ -2\ | = +2 \\
a(-1) &= |\ -1\ | = +1 \\
a(0) &= |\ 0\ | = 0 \\
a(+1) &= |\ +1\ | = +1 \\
a(+2) &= |\ +2\ | = +2 \\
a(+2.5) &= |\ +2.5\ | = +2.5
\end{aligned}$$

Absolute Value Definition

How is $|y|$ defined?

Well, you could always define it as the nonnegative square root of y^2:

$$|y| = \sqrt{y^2}$$

But here's another definition:

$$|y| = \begin{cases} -y, & \text{if } y \text{ is negative} \\ y, & \text{otherwise} \end{cases}$$

Absolute Value Implementation

$$|y| = \begin{cases} -y, & \text{if } y \text{ is negative} \\ y, & \text{otherwise} \end{cases}$$

Here's an implementation of absolute value in C:

```
if (y < 0) {
    absolute_value_of_y = -y;
} /* if (y < 0) */
else {
    absolute_value_of_y =  y;
} /* if (y < 0)...else */
```

What Does This Mean?

```
if (y < 0) {
    absolute_value_of_y = -y;
} /* if (y < 0) */
else {
    absolute_value_of_y =  y;
} /* if (y < 0)...else */
```

1. Evaluate the ***condition*** `(y < 0)`, which is a Boolean expression completely enclosed in parentheses, resulting in either true (1) or false (0).
2. In the event that the condition evaluates to true, then execute the statement inside the `if` clause.
3. Otherwise, execute the statement inside the `else` clause.

Branching with `if`

Branching is a way to ***select*** between possible sets of statements.

In C, the most common kind of branching is the ***`if` block***:

```
if (condition) {
    statement1;
    statement2;
    …
}
```

Example `if` Blocks

```
if (a > b) {
    printf("Wow, a is greater than b!\n");
} /* if (a > b) */

if (my_height < your_height) {
    shortest_height = my_height;
} /* if (my_height < your_height) */

if (drink_item_code == coffee_item_code) {
    drink_price = coffee_price;
} /* if (drink_item_code == coffee_item_code) */
```

`if` Condition

```
if (condition) {
    statement1;
    statement2;
    …
}
```

The condition is a Boolean expression completely enclosed in parentheses.

The ***condition*** is a **Boolean expression**, so it evaluates either to true (`1`) or to false (`0`).

The Boolean expression that constitutes the condition **MUST** be **completely enclosed in parentheses**.

`if` Block and Statement Terminators

```
if (condition) {
    statement1;
    statement2;
    …
}
```

The `if` statement is followed by a block open `{`
INSTEAD OF by a statement terminator (semicolon).

Statements inside the ***`if` clause*** are followed by statement terminators (semicolons) as appropriate, exactly the same as statements that aren't inside an `if` clause.

The block close `}` at the end of the `if` block
ISN'T followed by a statement terminator (semicolon).

`if` Block Indentation

```
if (condition) {
    statement1;
    statement2;
    …
}
```

Statements inside the `if` clause are indented additionally, beyond the indentation of the `if` statement and its associated block close.

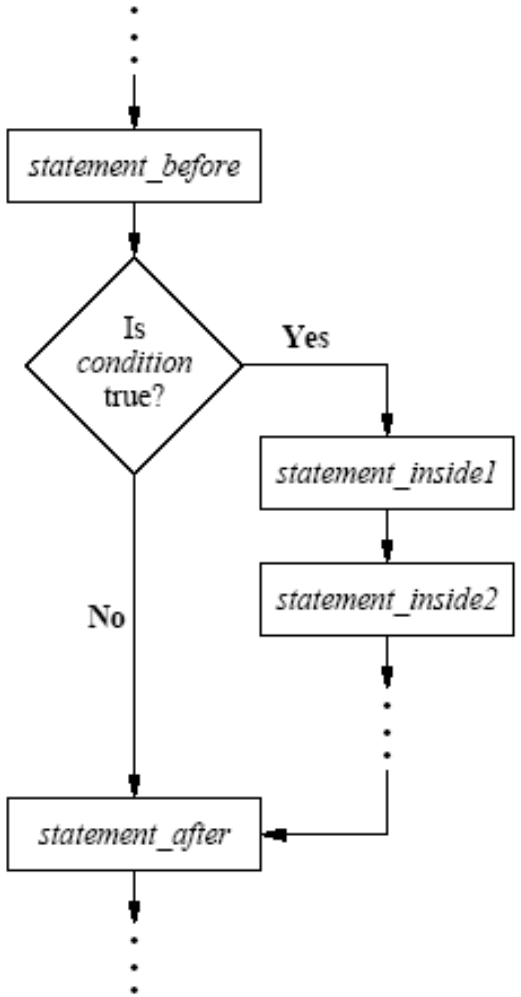
In CS1313, the statements inside the `if` clause are indented an additional **4 spaces** beyond the `if` statement and its associated block close.

In CS1313, you are **ABSOLUTELY FORBIDDEN** to use tabs for indenting in your source code.

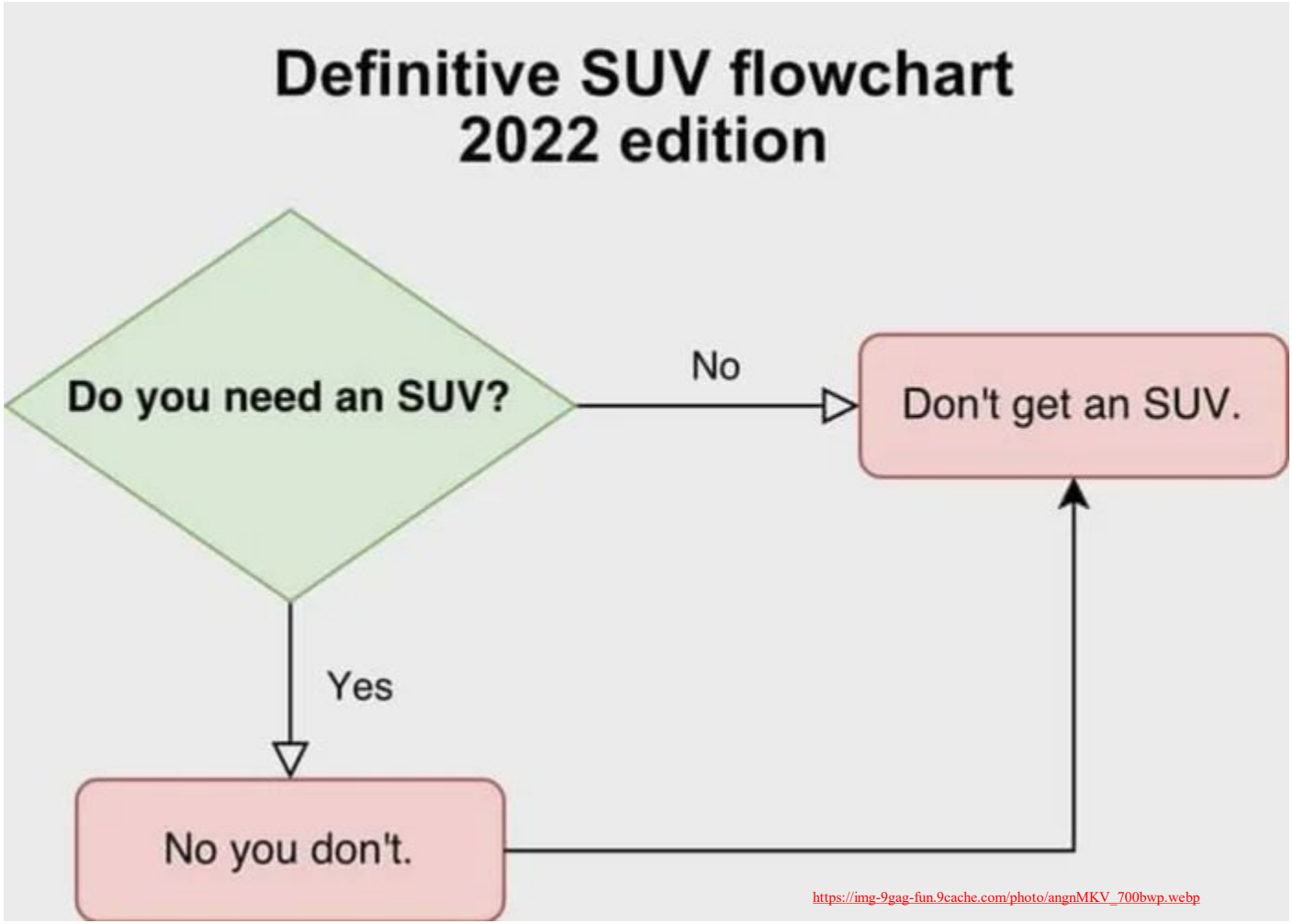
`if` Flowchart

```
statement_before;
if (condition) {
   statement_inside1;
   statement_inside2;
   ...
}
statement_after;
```

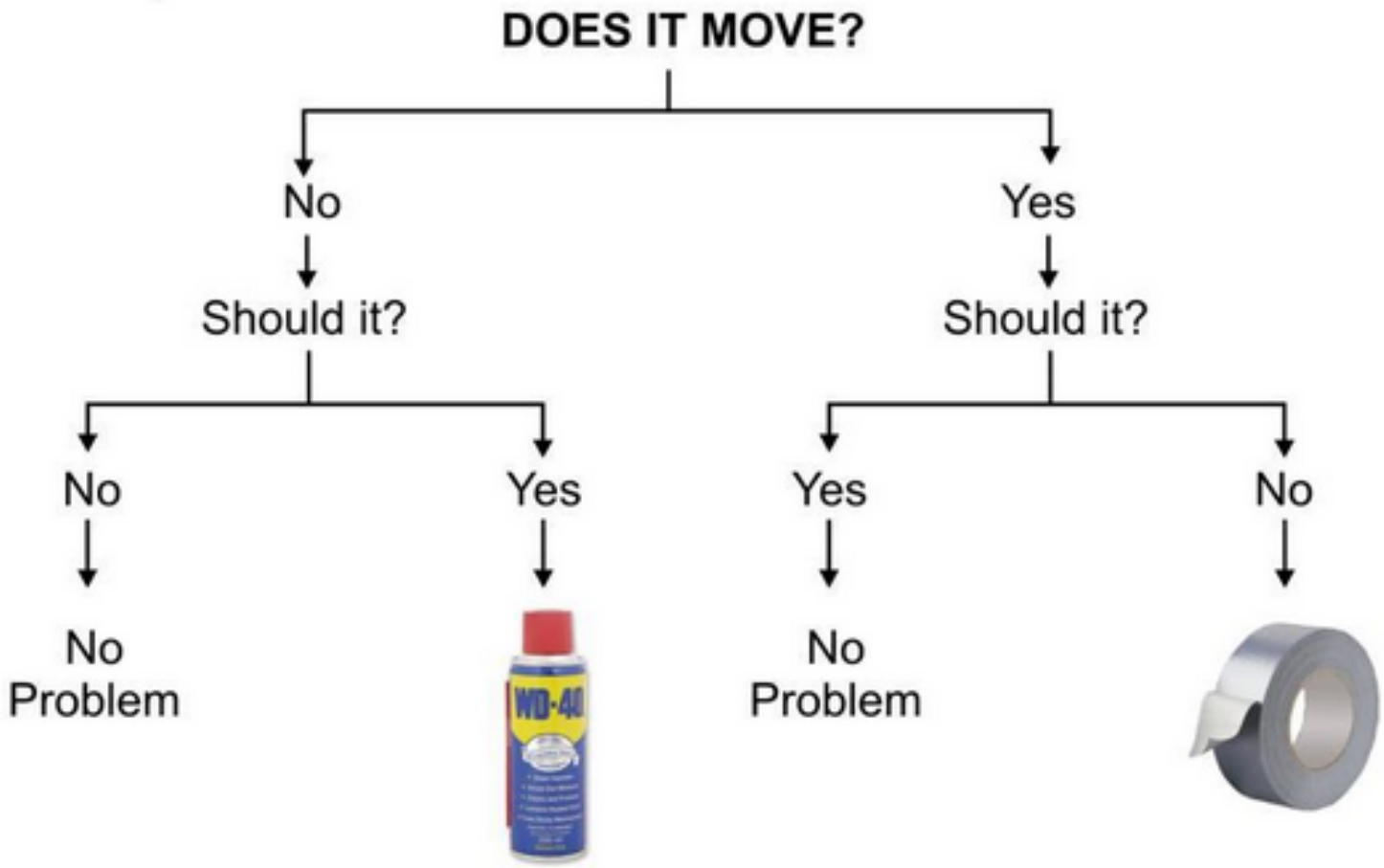
A **diamond** indicates a branch.



`if` Flowchart Example #1



`if` Flowchart Example #2



https://memelane.com/___image___?url=https%3A%2F%2Fi.imgur.com%2FGd5zPPi.png

The Meaning of `if` #1

In `my_number.c`, we saw something like this:

```
if ((users_number < minimum_number) ||
     (users_number > maximum_number)) {
    printf("Hey! That's not between %d and %d!\n",
        minimum_number, maximum_number);
} /* if ((users_number < minimum_number) || ... */
```

What does this mean?

The Meaning of `if` #2

```
if ((users_number < minimum_number) ||
    (users_number > maximum_number))  {
     printf("Hey! That's not between %d and %d!\n",
         minimum_number, maximum_number);
} /* if ((users_number < minimum_number) || ... */
```

Condition

First, the condition

```
((users_number < minimum_number) ||
 (users_number > maximum_number))
```

is evaluated, resulting in either true (`1`) or false (`0`).

AGAIN: The condition is a Boolean expression completely enclosed in parentheses.

The Meaning of `if` #3

```
if ((users_number < minimum_number) ||
    (users_number > maximum_number))  {
      printf("Hey! That's not between %d and %d!\n",
          minimum_number, maximum_number);
} /* if ((users_number < minimum_number) || ... */
```

Condition

Second, in the event that the condition evaluates to true (`1`), then the sequence of statement(s) **inside** the `if` clause – that is, between the block open of the `if` statement and the associated block close – are executed in order.

Otherwise, these statements are skipped.

The Meaning of `if` #4

```
if ((users_number < minimum_number) ||
    (users_number > maximum_number))  {
    printf("Hey! That's not between %d and %d!\n",
        minimum_number, maximum_number);
} /* if ((users_number < minimum_number) || ... */
```

Condition

Finally, regardless of whether the condition evaluates to true (`1`) or false (`0`), execution picks up at the next statement **immediately after** the block close of the `if` clause, and continues along from there.

The Meaning of `if` #5

```
if ((users_number < minimum_number) ||
    (users_number > maximum_number)) {
    printf("Hey! That's not between %d and %d!\n",
        minimum_number, maximum_number);
} /* if ((users_number < minimum_number) || ... */
```

Condition

In the event that the condition evaluates to **true** (`1`)

– that is, **IF** it's the case that `users_number` is less than `minimum_number` **OR** it's the case that `users_number` is greater than `maximum_number` – then the statement

```
printf("Hey! That's not between %d and %d!\n",
    minimum_number, maximum_number);
```

IS executed, in which case the output is:

```
Hey! That's not between 1 and 10!
```

The Meaning of `if` #6

```
if ((users_number < minimum_number) ||
    (users_number > maximum_number))  {
    printf("Hey! That's not between %d and %d!\n",
        minimum_number, maximum_number);
} /* if ((users_number < minimum_number) || ... */
```

Condition

On the other hand, in the event that `users_number` lies between `minimum_number` and `maximum_number` – that is, the condition evaluates to **false** (`0`) – then the `printf` statement is **skipped** (**NOT** executed), and therefore no output is produced by the `if` block.

`if` Example #1

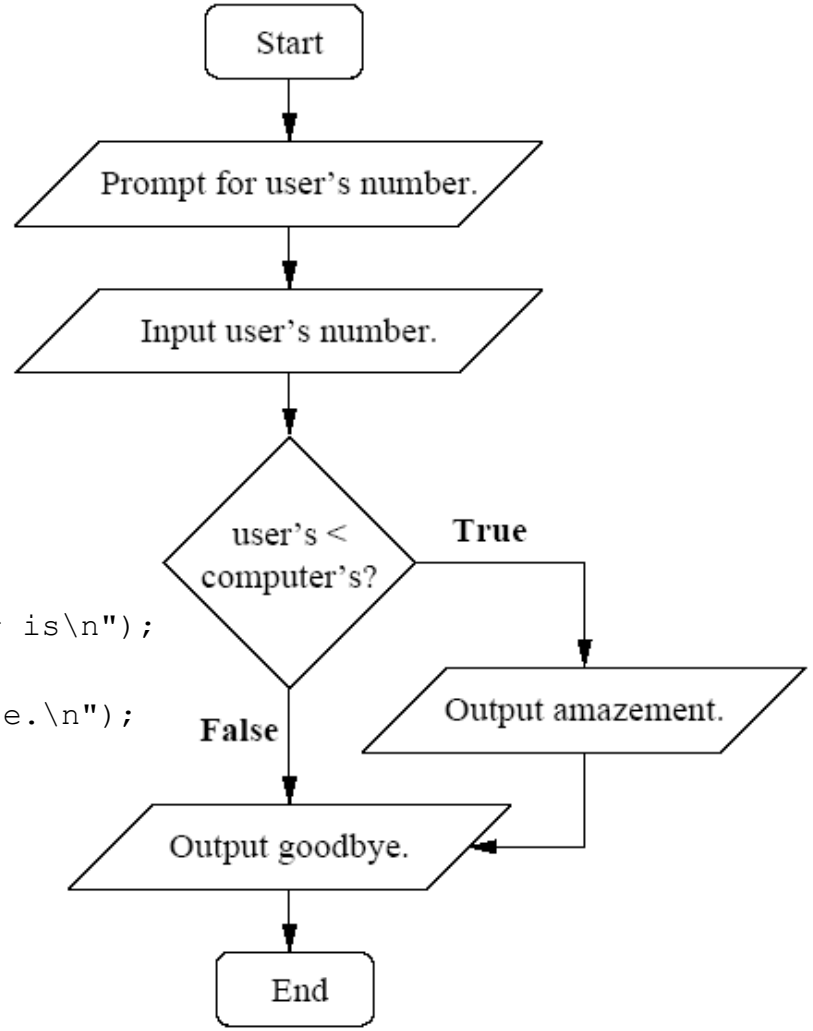
```
#include <stdio.h>

int main ()
{ /* main */
    const int computers_number = 5;
    int users_number;

    printf("Pick an integer:\n");
    scanf("%d", &users_number);
    if (users_number < computers_number) {
        printf("That's unbelievable! Your number is\n");
        printf("  less than mine!\n");
        printf("Well, okay, maybe it's believable.\n");
    } /* if (users_number < computers_number) */
    printf("And now I'm sick of you.\n");
    printf("Bye!\n");
} /* main */
```

`if` Example #2

```
% gcc -o isless isless.c
% isless
Pick an integer:
6
And now I'm sick of you.
Bye!
% isless
Pick an integer:
5
And now I'm sick of you.
Bye!
% isless
Pick an integer:
4
That's unbelievable! Your number is
  less than mine!
Well, okay, maybe it's believable.
And now I'm sick of you.
Bye!
```

`if` Example Flowchart

```
printf("Pick an integer:\n");
scanf("%d", &users_number);
if (users_number < computers_number) {
    printf("That's unbelievable! Your number is\n");
    printf("  less than mine!\n");
    printf("Well, okay, maybe it's believable.\n");
} /* if (users_number < computers_number) */
printf("And now I'm sick of you.\n");
printf("Bye!\n");
```



Block Open/Close Comments for `if` Block

```
if ((users_number < minimum_number) ||
    (users_number > maximum_number)) {
    printf("Hey! That's not between %d and %d!\n",
        minimum_number, maximum_number);
} /* if ((users_number < minimum_number) || ... */
```

NOTICE:

- The **block open** of this `if` block **doesn't** have a comment on the same line, because it's on the same line as the condition.
- The **block close** of this `if` block **does** have a comment on the same line, and that comment contains the `if` statement, or a shortened version of it, **EXCLUDING** its block close.

Boolean Expr Completely Parenthesized #1

```
if ((users_number < minimum_number) ||
    (users_number > maximum_number)) {
    printf("Hey! That's not between %d and %d!\n",
        minimum_number, maximum_number);
} /* if ((users_number < minimum_number) || ... */
```

Condition

The condition

```
((users_number < minimum_number) ||
 (users_number > maximum_number))
```

is a Boolean expression completely enclosed in parentheses.

How do we know this?

Boolean Expr Completely Parenthesized #2

```
if ((users_number < minimum_number) ||
    (users_number > maximum_number)) {
    printf("Hey! That's not between %d and %d!\n",
        minimum_number, maximum_number);
} /* if ((users_number < minimum_number) || ... */
```

Condition

First, this subexpression

```
(users_number < minimum_number)
```

is a Boolean expression, specifically a relational expression, so it evaluates to a Boolean value – true (`1`) or false (`0`).

Boolean Expr Completely Parenthesized #3

```
if ((users_number < minimum_number) ||
    (users_number > maximum_number)) {
    printf("Hey! That's not between %d and %d!\n",
        minimum_number, maximum_number);
} /* if ((users_number < minimum_number) || ... */
```

Condition

Second, this subexpression

```
(users_number > maximum_number)
```

is a Boolean expression, specifically a relational expression, so it evaluates to a Boolean value – true (`1`) or false (`0`).

Boolean Expr Completely Parenthesized #4

```
if ((users_number < minimum_number) ||
    (users_number > maximum_number)) {
    printf("Hey! That's not between %d and %d!\n",
        minimum_number, maximum_number);
} /* if ((users_number < minimum_number) || ... */
```

Condition

The condition

```
(users_number < minimum_number) ||
(users_number > maximum_number)
```

is a pair of Boolean subexpressions, specifically
relational expressions, joined by a Boolean operation, OR (`||`),
which in turn evaluates to a Boolean value – true (`1`) or false (`0`).

So the expression as a whole is a Boolean expression.

Boolean Expr Completely Parenthesized #5

```
if ((users_number < minimum_number) ||
    (users_number > maximum_number)) {
    printf("Hey! That's not between %d and %d!\n",
        minimum_number, maximum_number);
} /* if ((users_number < minimum_number) || ... */
```

Condition

The condition

```
((users_number < minimum_number) ||
 (users_number > maximum_number))
```

is a pair of Boolean subexpressions, specifically relational expressions, joined by a Boolean operation, OR (||), and the whole thing is enclosed in parentheses.

So: **The condition is a Boolean expression completely enclosed in parentheses.**

BAD Condition #1

```
if  (users_number < minimum_number)  ||
    (users_number > maximum_number)  {
      printf("Hey! That's not between %d and %d!\n",
          minimum_number, maximum_number);
} /* if ((users_number < minimum_number) || ... */
```

Condition

What if the condition

```
(users_number < minimum_number) ||
(users_number > maximum_number)
```

were a Boolean expression but were not completely enclosed in parentheses?

The compiler would treat this as an error!

It would be **WRONG WRONG WRONG.**

BAD BAD BAD Condition Example

```
% cat condnotenclosed.c
#include <stdio.h>

int main ()
{ /* main */
    const int minimum_number = 1;
    const int maximum_number = 10;
    int users_number = 0;

    if  (users_number < minimum_number) ||
        (users_number > maximum_number)  {
        printf("Hey! That's not between %d and %d!\n",
            minimum_number, maximum_number);
    } /* if  (users_number < minimum_number) || ... */
} /* main */
% gcc -o condnotenclosed condnotenclosed.c
condnotenclosed.c: In function main:
condnotenclosed.c:9: error: expected expression
  before || token
```

Notice that the compiler is **VERY UNHAPPY**.

GOOD Condition Example

```
% cat condenclosed.c
#include <stdio.h>

int main ()
{ /* main */
    const int minimum_number = 1;
    const int maximum_number = 10;
    int users_number = 0;

    if ((users_number < minimum_number) ||
        (users_number > maximum_number)) {
        printf("Hey! That's not between %d and %d!\n",
            minimum_number, maximum_number);
    } /* if ((users_number < minimum_number) || ... */
} /* main */
% gcc -o condenclosed condenclosed.c
% condenclosed
Hey! That's not between 1 and 10!
```

Notice that the compiler is now **HAPPY**, because the condition is now completely enclosed in parentheses.

Kinds of Statements Inside `if` Block

Between the `if` statement's block open and
the associated block close,
there can be **any kind** of **executable** statements,
and **any number** of them.

For example:

- `printf` statements;
- `scanf` statements;
- assignment statements;
- `if` blocks.

There are several other kinds of executable statements
that can occur inside an `if` block that we haven't learned yet,
some of which we'll learn later in the semester.

Statements Inside `if` Block

In the event that the `if` condition evaluates to true (`1`), then the statements inside the `if` block will be executed one by one, in the order in which they appear in the `if` block.

No Declarations Inside `if` Block

Notice that an `if` block **SHOULDN'T** contain declaration statements, because the `if` statement is an executable statement, and **ALL** declarations **MUST** come before **ANY** executable statements.

Absolute Value Example #1

```
% cat absval.c
#include <stdio.h>

int main ()
{ /* main */
    const int minimum_for_not_negating = 0;
    float input_value, output_value;

    printf("I'm going to calculate the absolute\n");
    printf("  value of a value that you input.\n");
    printf("Please input the value.\n");
    scanf("%f", &input_value);
    if (input_value < minimum_for_not_negating) {
        output_value = -input_value;
    } /* if (input_value < ...) */
    else {
        output_value =  input_value;
    } /* if (input_value < ...)...else */
    printf("The absolute value of %f is %f.\n",
        input_value, output_value);
} /* main */
```

Absolute Value Example #2

```
% gcc -o absval absval.c
% absval
I'm going to calculate the absolute
  value of a value that you input.
Please input the value.
5
The absolute value of 5.000000 is 5.000000.
% absval
I'm going to calculate the absolute
  value of a value that you input.
Please input the value.
-5
The absolute value of -5.000000 is 5.000000.
```

A More Complicated `if` Example #1

```
#include <stdio.h>
#include <stdlib.h>

int main ()
{ /* main */
    const int int_code                  =  1;
    const int float_code                =  2;
    const int minimum_for_not_negating =  0;
    const int program_faìlure_code = -1;
    const int program_success_code =  0;
    float float_input_value, float_output_value;
    int   int_input_value,   ínt_output_value;
    int   data_type_code;

    printf("I'm going to calculate the absolute value\n");
    printf("  of a number that you input.\n");
```

A More Complicated `if` Example #2

```
printf("Would you like to input an int or a float?\n");
printf("  (Enter %d for an int or %d for a float.)\n",
    int_code, float_code);
scanf("%d", &data_type_code);
if ((data_type_code != int_code) &&
    (data_type_code != float_code)) {
    printf("ERROR: I don't recognize data type code %d.\n",
        data_type_code);
    exit(program_failure_code);
} /* if ((data_type_code != int_code) ... */
if (data_type_code == int_code) {
    printf("What is the int value?\n");
    scanf("%d", &int_input_value);
} /* if (data_type_code == int_code) */
else if (data_type_code == float_code) {
    printf("What is the float value?\n");
    scanf("%f", &float_input_value);
} /* if (data_type_code == float_code) */
```

Idiotproofing

A More Complicated `if` Example #3

```
if (data_type_code == int_code) {
    if (ìnt_input_value < minimum_for_not_negating) {
        int_output_value = -int_input_value;
    } /* if (int_input_value < ...) */
    else {
        int_output_value = +int_input_value;
    } /* if (int_input_value < ...)...else */
} /* if (data_type_code == int_code) */
else if (data_type_code == float_code) {
    if (float_ínput_value < minimum_for_not_negating) {
        float_output_value = -float_ìnput_value;
    } /* if (float_input_value < ...) */
    else {
        float_output_value = +float_input_value;
    } /* if (float_input_value < ...)...else */
} /* if (data_type_code == float_code) */
```

A More Complicated `if` Example #4

```
    if (data_type_code == int_code) {
        printf("The absolute value of %d is %d.\n",
            int_ìnput_value, int_output_value);
    } /* if (data_type_code == int_code) */
    else if (data_type_code == float_code) {
        printf("The absolute value of %f is %f.\n",
            float_ínput_value, float_output_value);
    } /* if (data_type_code == float_code) */

    return program_success_code;

} /* main */
```

A More Complicated `if` Example Runs #1

```
% gcc -o absvalbytype absvalbytype.c
% absvalbytype
I'm going to calculate the absolute value
  of a number that you input.
Would you like to input an int or a float?
  (Enter 1 for an int or 2 for a float.)
0
ERROR: I don't recognize data type code 0.
```

A More Complicated `if` Example Runs #2

```
% absvalbytype
I'm going to calculate the absolute value
  of a number that you input.
Would you like to input an int or a float?
  (Enter 1 for an int or 2 for a float.)
1
What is the int value?
5
The absolute value of 5 is 5.
% absvalbytype
I'm going to calculate the absolute value
  of a number that you input.
Would you like to input an int or a float?
  (Enter 1 for an int or 2 for a float.)
1
What is the int value?
-5
The absolute value of -5 is 5.
```

A More Complicated `if` Example Runs #3

```
% absvalbytype
I'm going to calculate the absolute value
  of a number that you input.
Would you like to input an int or a float?
  (Enter 1 for an int or 2 for a float.)
2
What is the float value?
5.5
The absolute value of 5.500000 is 5.500000.
% absvalbytype
I'm going to calculate the absolute value
  of a number that you input.
Would you like to input an int or a float?
  (Enter 1 for an int or 2 for a float.)
2
What is the float value?
-5.5
The absolute value of -5.500000 is 5.500000.
```

Compound Statement a.k.a. Block #1

A ***compound statement*** is a sequence of statements, with a well-defined beginning and a well-defined end, to be executed, in order, under certain circumstances.

An `if` block is a compound statement. We'll see others later.

Although an `if` block is actually a sequence of statements, we can think of it as a single "super" statement in some contexts.

Compound statements are also known as ***blocks***. Thus, we speak of an **`if` block**.

Compound Statement a.k.a. Block #2

In C, a compound statement, also known as a block, is delimited by curly braces.

That is, a compound statement (block):

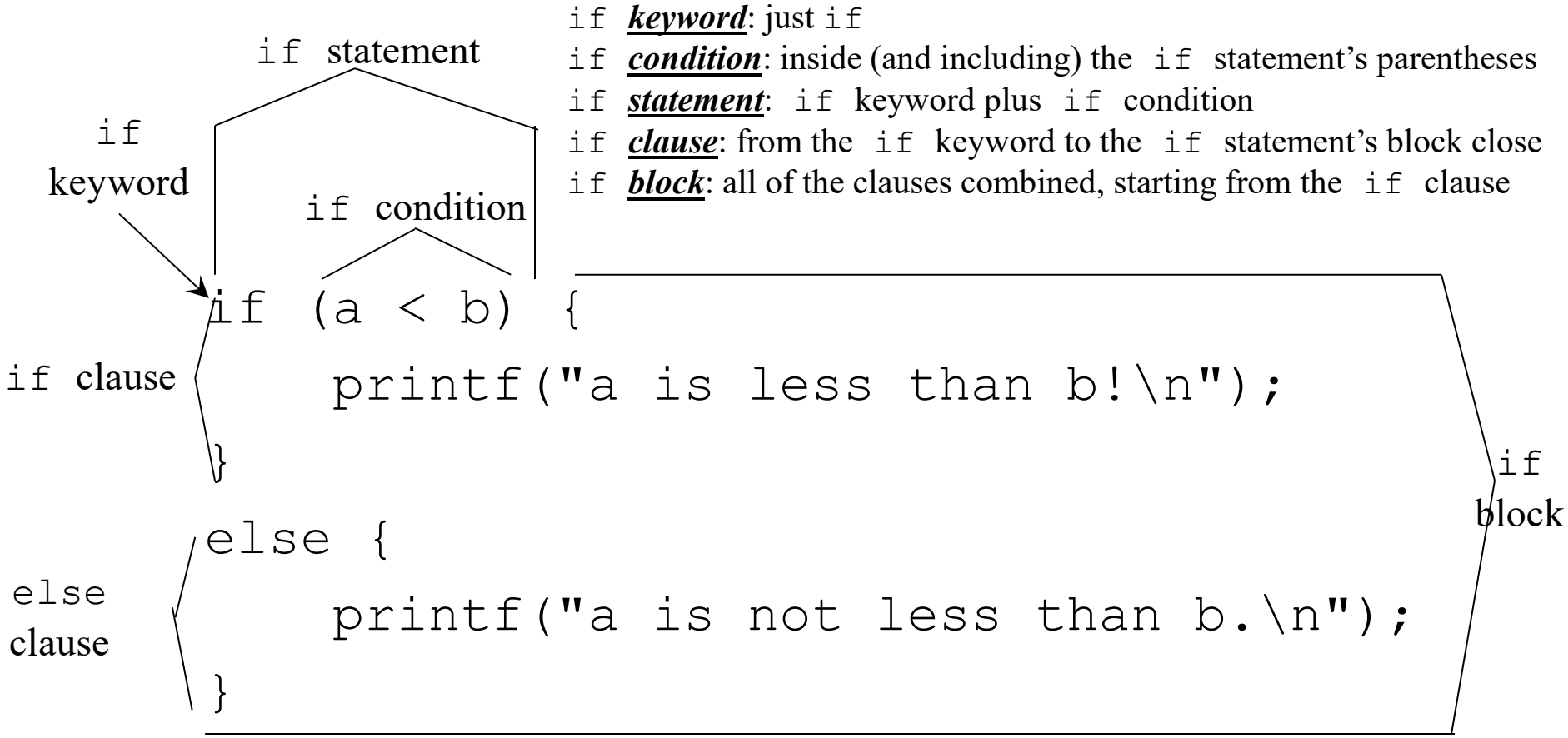
- begins with a **block open**

```
{
```

- ends with a **block close**

```
}
```

`if` Keyword, Condition, Statement, Clause, Block



`if` ***keyword***: just `if`
`if` ***condition***: inside (and including) the `if` statement's parentheses
`if` ***statement***: `if` keyword plus `if` condition
`if` ***clause***: from the `if` keyword to the `if` statement's block close
`if` ***block***: all of the clauses combined, starting from the `if` clause