Branching

Outline

1. *Boolean* Type
2. Relational and Logical Operators
3. Operator Precedence
4. Selection Control Structure/Branching (**`if-else`**)

1. *Boolean* Type

- A *Boolean* data type is a data type that has two values (usually *true* and *false*).
- C language does not have a Boolean data type.
- C language treats **zero** as false and **non-zero** as true.
- A Boolean expression in C language evaluates to either 0 (false) or 1 (true).

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 `0` (false) or `1` (true).

- Suppose **x** and **y** are declared as
  `int x = 0, y = 100;`

Observe with care:
= and **==** are different!

```
x > 10                  ➔   0

y >= (x + 100)          ➔   1

100 == (x + y)          ➔   1
```

2.1. Comparison Example

- What is the output of the following program?

```c
#include <stdio.h>

int main()
{
    int x=0;
    int y=100;

    printf("%d\n", x > 10);
    printf("%d\n", y >=(x+100));
    printf("%d\n", 100 == (x+y));

    return 0;
}
```

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 a **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*", one can write

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

2.2.2. Evaluating Logical **OR** operator (||)

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

- If **any** of the operands is true, the result is true.

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

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

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)
```

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.

- e.g.,

  5 && 5 ➔ "true **AND** true" ➔ 1 (*5 treated as true*)

  !5 ➔ **NOT** true ➔ 0

  !!5 ➔ **NOT NOT** true ➔ 1

3. Operator Precedence

- We have learned about operator precedence last time
- Do you remember what is
  - Operator precedence?
  - Operator associativity?

3. Operator Precedence

- *Operator precedence*
  - Among different operators, *operator precedence* tells us which operator(s) should be applied first.

- *Operator associativity*
  - Among operators with the same precedence, *operator associativity* tells us whether the left-most or the right-most operator should be applied first.

3. Operator Precedence

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	
`=` `+=` `-=` `*=` `/=` etc.	right to left	Lowest

3.1. Operator Precedence Practice

- 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) `5 < x < 10` **(ATTENTION! This expression does not evaluate as in Math!)**

3.1. Operator Precedence Practice

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

```c
#include <stdio.h>

int main()
{
    int x=0;

    printf("%d\n", 5 < x);
    printf("%d\n", (5 < x) < 10);
    printf("%d\n", 0 < 10);
    printf("%d\n", 5 < x < 10);
    return 0;
}
```

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

What should be the right expression to use in C?

3.2. Writing Logical Expressions

- Assuming **x** and **y** are integer variables, how do we write the following conditions in C expressions?
- "*x is a number between 0 and 10* (exclusive)":
  - In other words, x is greater than 0, and x is less than 10

```
x > 0 && x < 10
```

- "*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)
```

3.2. Writing Logical Expressions

Challenge: How about the following?

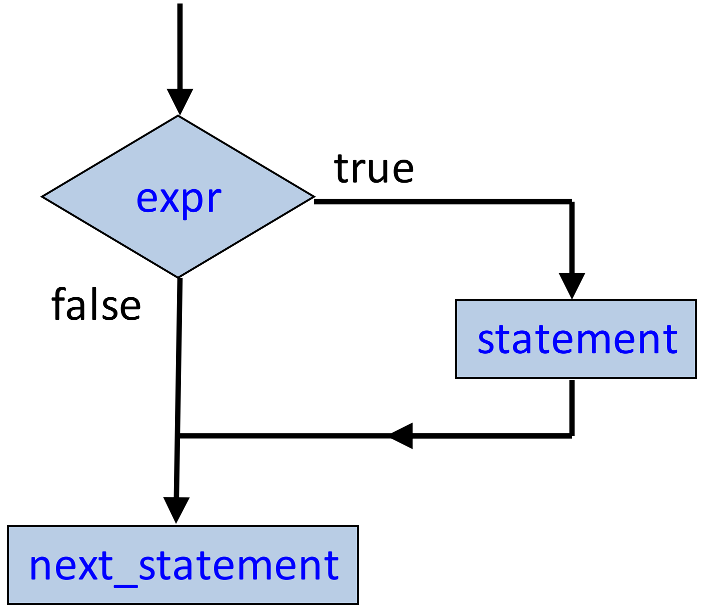
a) "*Exactly one of x and y is zero.*"
   - In other words, when x is zero, y is not zero, or when y is zero, x is not zero.

b) "*x, y, and z are identical.*"

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

4. `if` Statement (*Syntax*)

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



- 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`**.

4.1. `if` Statement (*Example*)

```c
int score;
printf("Please enter your score: ");
scanf("%d", &score);

if (score >= 60)
    printf("Passed!\n");

if (score < 60)
    printf("Failed!\n");

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.
```

4.2. Indentation

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

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

same as

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

Indentation – adding spaces at the beginning of a line to align codes.

Indenting codes does not affect a program; it only makes the codes easier to read.

- How do we conditionally execute *multiple* statements then?

4.3. Compound Statement

```c
if (score >= 60) {
    printf("You have passed!\n");
    printf("Congratulations!\n");
}

printf("Your score is %d\n", score);

```

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

- **{ … }** groups multiple statements into ONE *compound statement.*
- Semicolon (**;**) not needed after **{ … }**

4.4. 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

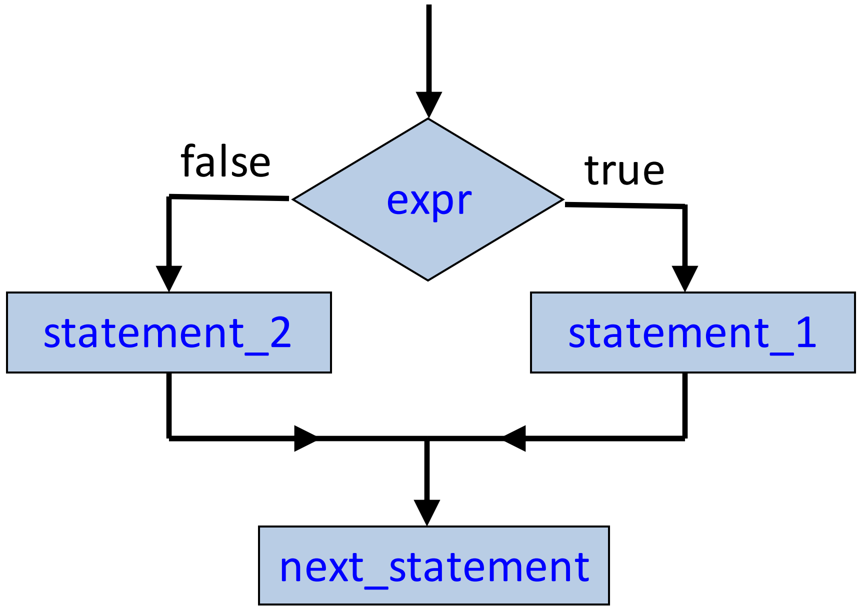
```
if (a != 0);
   some_statement;
```

interpreted as

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

4.5. `if-else` Statement (*Syntax*)

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



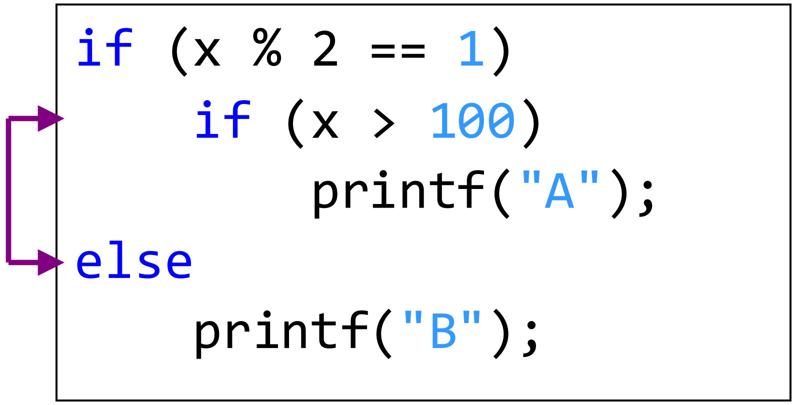
- Allows us to conditionally perform *either* one of two tasks, but never both

4.5.1. `if-else` Statement (*Example*)

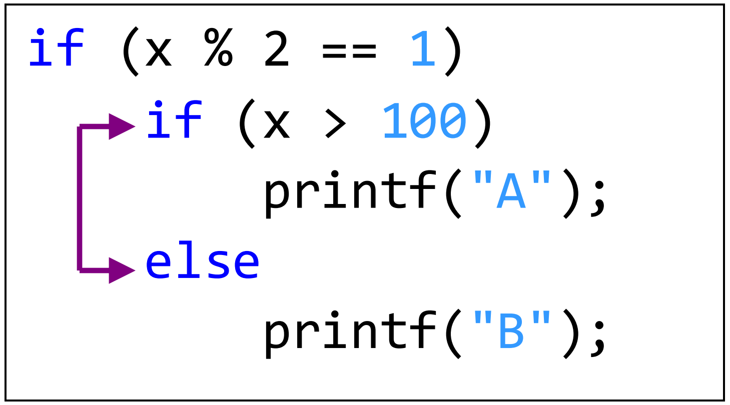
```c
int score;
printf("Please enter your score: ");
scanf("%d", &score);

if (score >= 60)
    printf("Passed!\n");
else                           // Replaces "if (score < 60)"
    printf("Failed!\n");

printf("Your score is %d.\n", score);
```

- If "`score >= 60`" is *true*, then "`score < 60`" must be *false*, and vice versa.

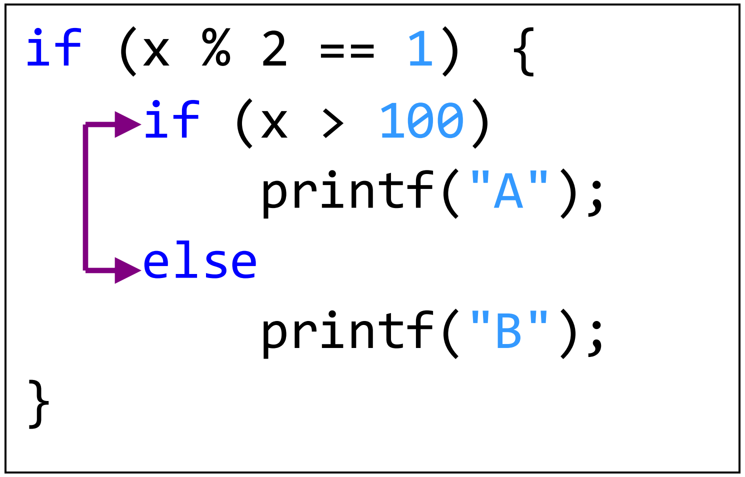
4.6. How does `else` pair with `if`?



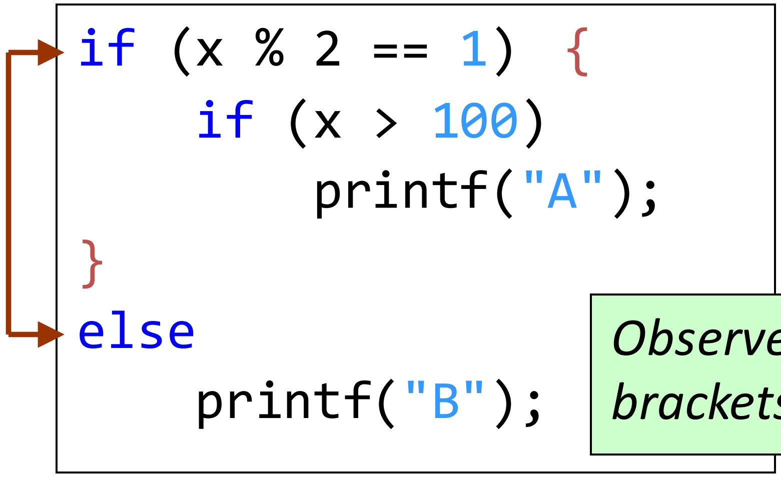
```
if (x % 2 == 1)
    if (x > 100)
        printf("A");
else
    printf("B");
```



```
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 {…}!

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

4.7. Conditionally performing 1 of N tasks

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

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

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

```c
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, that means:

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

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

```c
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)

4.7. 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 (no matter how long/compound it is) is treated as one statement. Therefore, the `{ … }` surrounding the inner `if-else` statements are optional.

4.7. 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.

Summary

- Relational Operators and Logical Operators

  `==` `!=` `<` `>` `<=` `>=` `!` `&&` `||`

- `if-else` Statement

- Nested `if-else` Statement

Reading Assignment

- C: How to Program, 8th ed, Deitel and Deitel
- Chapter 3 Structured Program Development in C
  - Sections 3.4, 3.5, 3.6
- Chapter 4 C Program Control
  - Sections 4.10, 4.11