(4-2) Selection Structures in C
H&K Chapter 4

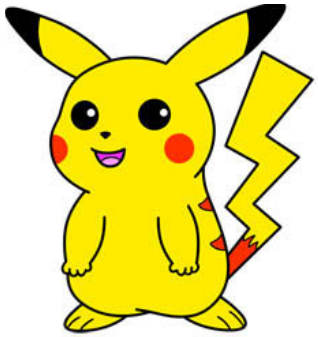
Instructor – Beiyu Lin

CptS 121 (May 14th, 2019)

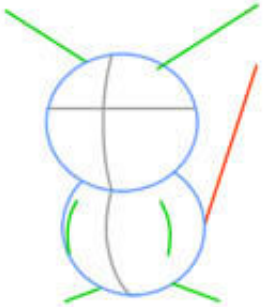
Washington State University

Agile Methodology – Software Development

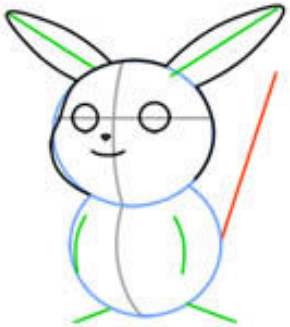
Goal!



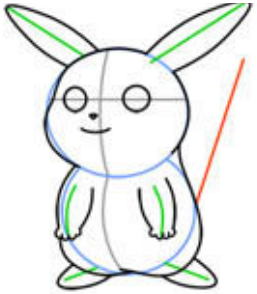
Sprint 0



Sprint 1

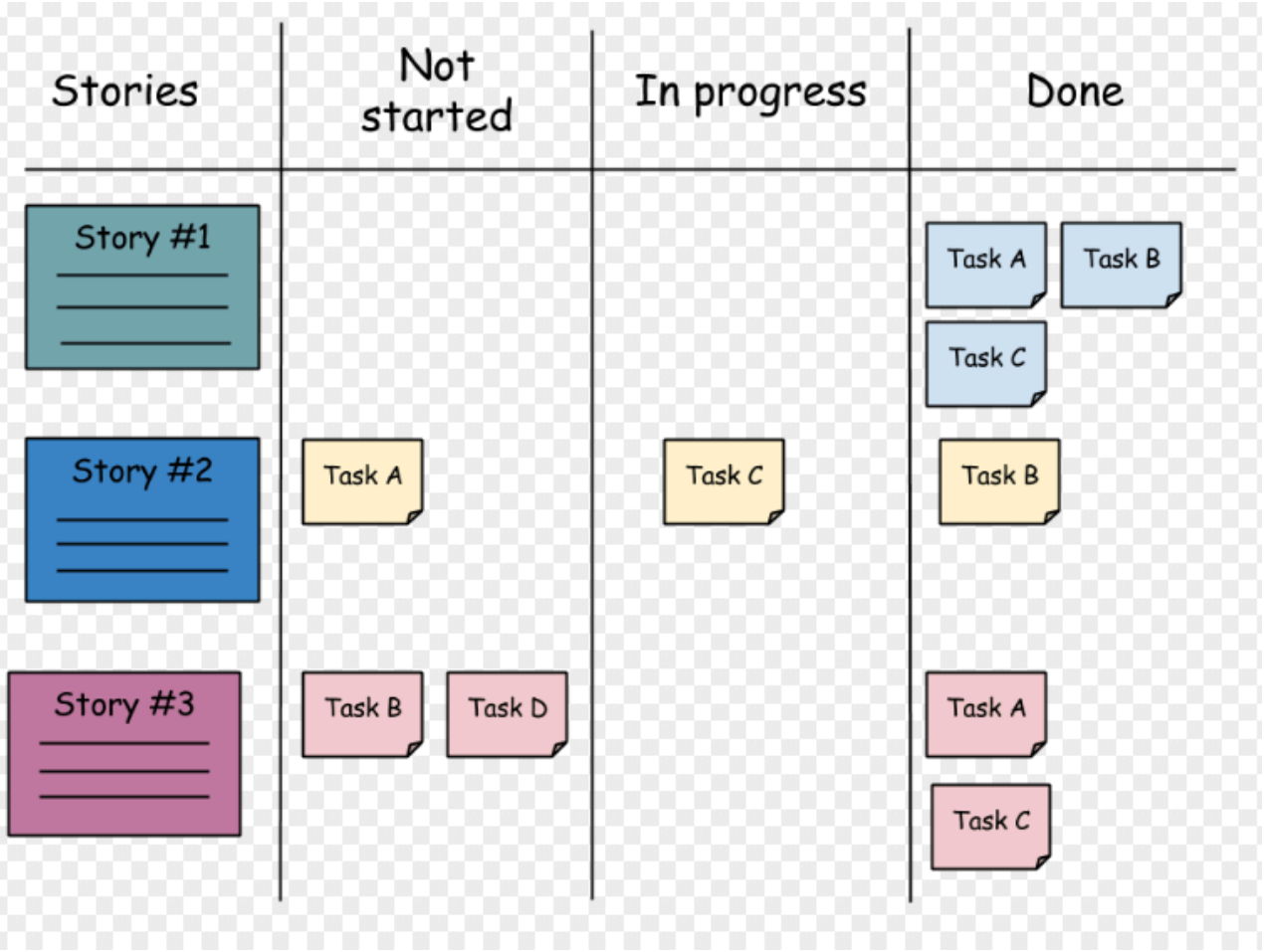


Sprint 2



Sprint 3

Agile Methodology – Sprint Planning



Control Structures

- Recall that algorithms are composed of three different kinds of statements:
  - Sequence: the ability to execute a series of instructions, one after the other.
  - Conditional: the ability to execute an instruction contingent upon some condition.
  - Iteration: the ability to execute one or more instructions repeatedly.
- This week, we'll learn about conditionals: the ability to execute some code IF some condition is true.

Conditions (1)

- Conditional statements rely on a Boolean *condition*, which evaluates to either *true* or *false*
- In C, the true and false values are actually represented numerically:
  - False: 0
  - True: any number except 0 (usually 1)
- Relational operators are used to build Boolean conditions:
  - < (less than), > (greater than), <= (less than or equal to), >= (greater than or equal to), == (equal to), != (not equal to)

Conditions (2)

- Examples
  - Assume `x = 3, y = 4, max = 100, min = 0, and ch = 'c'`
  - Then what do the following evaluate to?

```
x <= 0
x == y
max >= min
ch < 'a'
max
min != 0
max == 99 + 1
min - 50 < 0
```

Conditions (3)

- Logical operators
  - We can combine relational operators with logical operators to construct general Boolean expressions in C:
    - AND: `&&` (A single '&' is different.)
    - OR: `||` (A single '|' is different.)
    - NOT: `!`
  - Examples
    - Assume that `temp = 50`, `MAX_TEMP = 90`, `precip = 2.0`, `num_votes = 20`, `votes_needed = 20`, and `elected = 0`;
    - Then evaluate these:

```
(temp < MAX_TEMP) && (precip > 0)
(num_votes >= votes_needed) || (!elected)
```

Conditions (4)

- Operator precedence
  - Just like numeric operators (+, -, /, *), logical operators have precedence rules that determine order of evaluation
  - From highest to lowest, the precedences are as follows: [Most are left-to-right; but not assignment]

    function calls (highest)

    `!, +, -, &` (unary operators)

    `*, /, %`

    `+, -`

    `<, <=, >=, >`

    `==, !=`

    `&&`

    `||`

    `=` (assignment) (lowest)

Conditions (5)

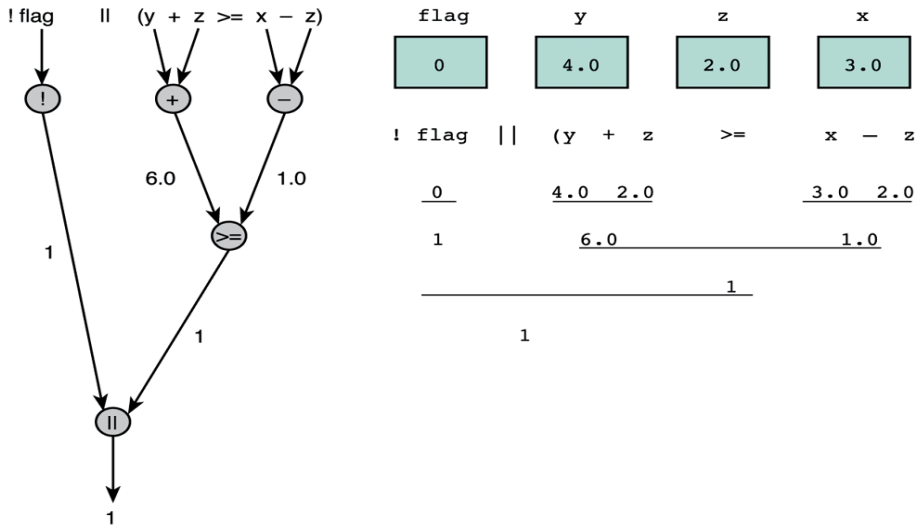
- Operator precedence (cont.)
  - When in doubt, parenthesize!
    - How will the expression

```
x + y < z - a
```

      evaluate?
    - Answer: `(x + y) < (z - a)`
    - Nonetheless, it's a good idea to parenthesize as above in order to make the order of evaluation clear
    - However, over-parenthesizing may decrease code readability

Conditions (6)

- Operator precedence (cont.)
  - Consider the expression
    `!flag || (y + z >= x - z)`
  - Here's how it's evaluated, assuming
    `flag = 0, y = 4.0, z = 2.0,` and `x = 3.0`:



Conditions (7)

- Short-circuit evaluation
  - Notice that, in case of `&&` (and), if the first part of expression is false, the entire expression must be false
    - Example: `(5 < 3) && (4 > 3)`
  - Likewise, in case of `||` (or), if the first part of expression is true, the entire expression must be true
    - Example: `(4 > 2) || (2 > 3)`
  - In these two cases, C *short-circuits* evaluation
    - Evaluation stops after first part of expression is evaluated

Conditions (8)

- Logical assignment
  - It is possible to assign a logical expression to an `int` variable
  - Example:

```
int temp; /* input, the current temperature */
int below_freezing /* temperature state */
scanf("%d",&temp);
below_freezing = (temp < 32); /* Sets temperature
                                 state variable */
```

  - The `below_freezing` variable can then be involved in logical expressions, for example:

```
below_freezing && (dew_point < 32)
```


Conditions (9)

- Complementing conditions
  - The complement of a condition can be obtained by applying the ! operator
  - Example: The complement of `temp > 32` is `!(temp > 32)`, which can also be written as `temp <= 32`
  - Example: The complement of temp == 32 is `!(temp == 32)`, which can also be written as `(temp != 32)`

Conditions (10)

- Complementing conditions (cont.)
  - Use DeMorgan's laws to complement compound logical expressions:
    - The complement of X && Y is !X || !Y
    - The complement of X || Y is !X && !Y
    - Example:

      ```
      (temp > 32) && (skies == 'S' || skies == 'P')
      ```

      would be complemented as follows:

      ```
      (temp <= 32) || (skies != 'S' && skies != 'P')
      ```

      Assuming that 'S' stands for sunny and 'P' stands for partly cloudy, the original condition is true if the temperature is above freezing and the skies are either sunny or partly cloudy. The complemented condition is true if the temperature is at or below freezing, and the skies are neither sunny nor partly cloudy.

The `if` Statement

- The if statement supports conditional execution in C:

```
if <test>
{
  <body>
}
```

- `<test>` must be an expression that can be evaluated to either true or false (non-zero or zero)
- `<body>` is one or more C statements.
- Although it is not required in cases where body has exactly one statement, it is better style to always enclose `<body>` in curly braces

The `if-else` statement

- C also defines an ‘if-else’ statement:

```
if <test>
{
     <body-if-test-is-true>
}
else
{
     <body-if-test-is-false>
}
```

- Note that only one of the two `<body>` blocks can be executed each time through this code. In other words, they are “mutually exclusive”.
- Also note `else` has no `<test>` condition.

Modeling

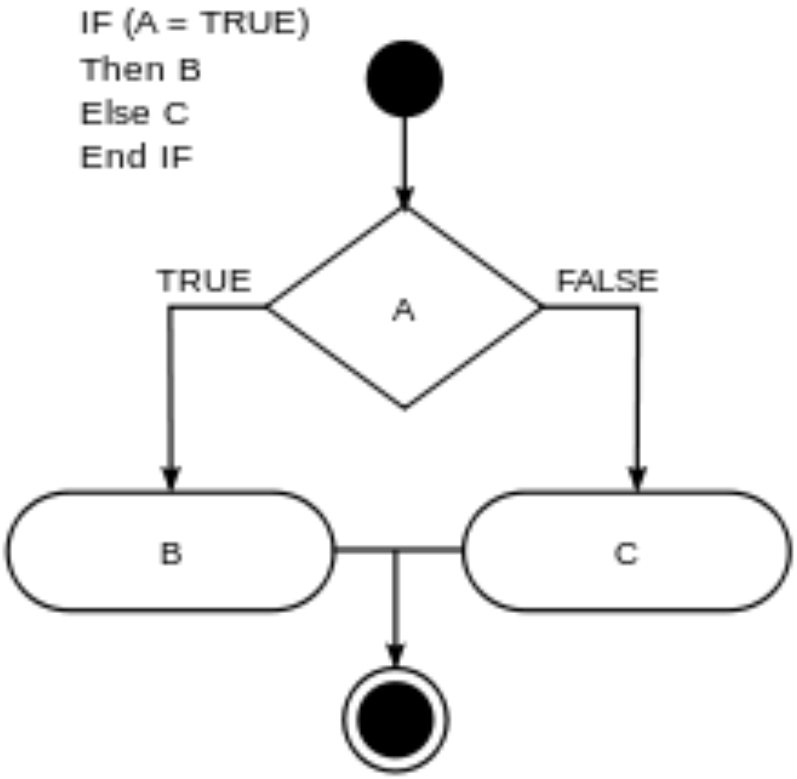
- A large part of software engineering is modeling
- We will define modeling as the practice of abstracting details to make software development more comprehendible
- Helps with designing and quality
- Models are usually visual/diagrams
- A flowchart is a model representing a process

Flowcharts (1)

- Diagram that uses boxes and arrows to show the step-by-step execution of a control structure
- Diamond shapes represent decisions
- Always one path into a decision and two going out
  - The paths are a result of false and true conditions
- We should construct flowcharts as part of our design of algorithms before implementation

Flowcharts (2)

- Below is an example of a flowchart:



Flowcharts (3)

- What does the associated C code look like for the previous flowchart?

```
…
    if (temperature > 32)
    {
            printf (“It’s warm out!\n”);
    }
    else
    {
            printf (“It’s freezing out!\n”);
    }
```


Flowcharts (3)

- What does the associated C code look like for the previous flowchart?

```
if(boolean_expression 1) {
   /* Executes when the boolean expression 1 is true */
} else if( boolean_expression 2) {
   /* Executes when the boolean expression 2 is true */
} else if( boolean_expression 3) {
   /* Executes when the boolean expression 3 is true */
} else {
   /* executes when the none of the above condition is true */
}
```


You Try It

- What does this do? (Careful!) The condition is always evaluated to 3. **We want x == 3!**

```
int x = 0;
if (x = 3)
{
  printf("x is small\n");
}
```

- What does this do?

```
int x = y = z = 0;
y = y + 4;
z = z * x;
if (z > y)
{
  printf("Z: %d.\n", z + 1);
}
else
{
  printf("X: %d.\n", x - 1);
}
```


Example Application (1)

- Build a payroll system that
  - prompts for employee pay rate,
  - prompts for first week's work hours,
  - prompts for second week's work hours,
  - prompts for third week's work hours, and
  - displays the employee's paycheck amount
  - Special conditions:
    - Overtime (> 40 hours) is time-and-a-half.
    - Issue warning whenever prior two weeks' overtime exceeds 30 hours.

Example Application (2)

- Example execution

```
Enter hourly pay rate: 8.00
Enter week1 hours: 40
   Paycheck is: 320.00
Enter week2 hours: 60
   Paycheck is: 560.00
Enter week3 hours: 60
   Paycheck is: 560.00
   Warning! Overtime is: 40
```


Example Application (3)

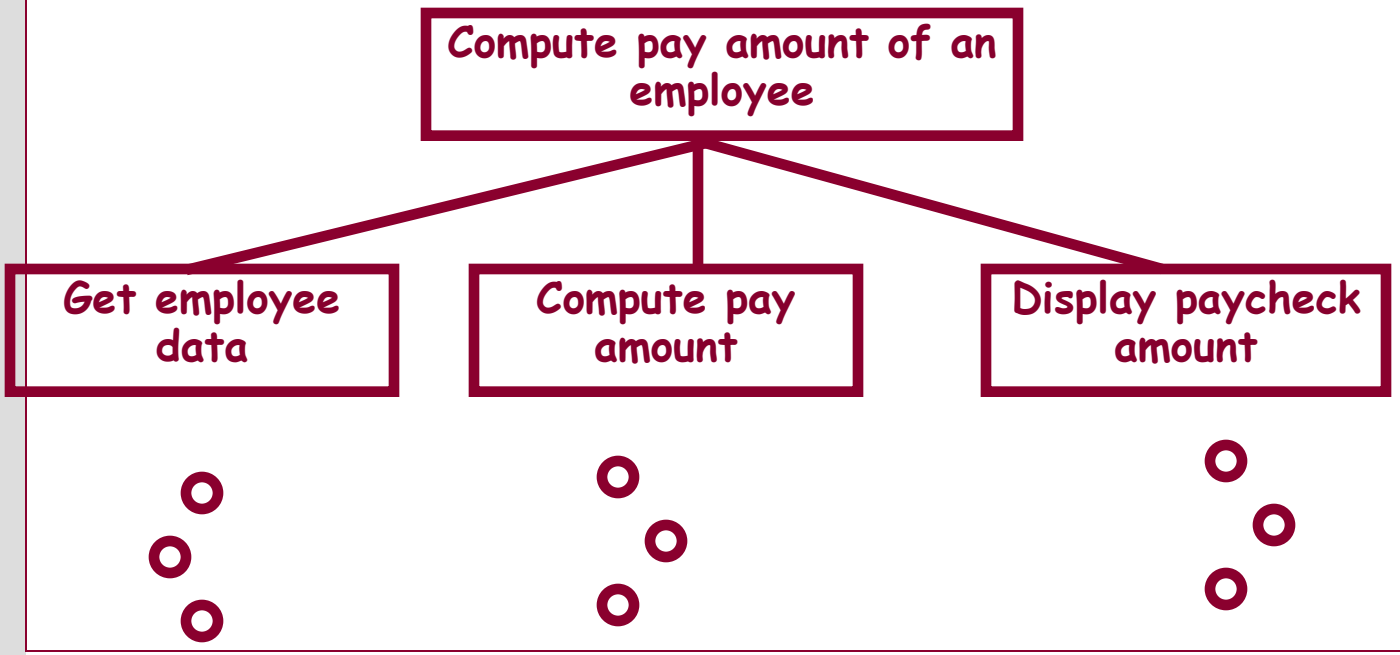
- Analysis/Data Requirements
  - Inputs:
    - `hourly_rate (double)`
    - `week1_hours (double)`
    - `week2_hours (double)`
    - `week3_hours (double)`
  - Outputs
    - `Paycheck_amount`
    - Warning if two consecutive weeks' overtime hours exceed 30

Example Application (4)

- Design
  - Initial algorithm
    - Get employee pay rate and weekly hours
    - Calculate paycheck amount
    - Display paycheck amount
    - Display warning
  - Refined algorithm
    - Get employee pay rate and weekly hours
    - For each week of pay
      - Calculate paycheck amount
        - If hours > 40
overtime = hours – 40
else
overtime = 0
        - If overtime = 0 then
pay = pay_rate * hours
else
pay = pay_rate * 40 + (1.5 * pay_rate * overtime)
      - Display warning
        - If overtime for two consecutive weeks > 30 then display warning

Example Application (5)

- Structure chart




```
get_payrate          compute_paycheck_amount()  display_paycheck_amount()
get_hours            compute_overtime()         display_warning()
```


Example Application (6)

```
/*** Computes the paycheck amount of an employee. ***/
#include <stdio.h>                    /* printf, scanf defs */

/*** Function prototypes ***/
double get_employee_payrate(void); /* prompts and reads in payrate */
double get_employee_hours(int)     /* prompts and reads in weekly hours */
double compute_paycheck_amount(double, double); /* computes weekly pay */
double compute_overtime(double);   /* computes hours of overtime */
void display_paycheck_amount(double); /* displays the pay amount */
void display_warning(double);      /* displays a warning message if */
                                   /* overtime exceeds 30.0 hours */
int main(void)
{
     char *name;
     double pay_rate, hours1, hours2, hours3, overtime1, overtime2, overtime3,
            pay1, pay2, pay3;
     overtime1 = overtime2 = overtime3 = 0.0;
     pay_rate =  get_payrate();
     hours1 = get_hours(1);
     pay1 = compute_paycheck_amount(hours1,pay_rate);
     overtime1 = compute_overtime(hours1);
     display_paycheck_amount(pay1);
```


Example Application (7)

```
    hours2 = get_hours(2);
    pay2 = compute_weekly_pay(hours2,pay_rate);
    overtime2 = compute_overtime(hours2);
    display_paycheck_amount(pay2);
    display_warning(overtime1 + overtime2); // displays warning only
                                            // if overtime exceeds 30
    hours3 = get_employee_hours(3);
    pay3 = compute_weekly_pay(hours3,pay_rate);
    overtime3 = compute_overtime(hours3);
    display_paycheck_amount(pay3);
    display_warning(overtime2 + overtime3); // displays warning only
                                            // if overtime exceeds 30
}

/*** Your function definitions go here ... ***/
```


Common Mistakes with `if` Statements

- Using = (assignment) instead of == (logical equality)
  - The compiler will NOT catch this mistake!
- Using `if-else` when `if-if` should be used
  - Remember `else` does not have an explicit condition associated with it
- Using logical AND (&&) instead of logical OR (||) and vice versa
  - Also single & and | will perform bitwise operations!

Next Lecture…

- Nested `if-else` statements
- `switch` statements
- Another example

References

- J.R. Hanly & E.B. Koffman, *Problem Solving and Program Design in C (8th Ed.)*, Addison-Wesley, 2016
- P.J. Deitel & H.M. Deitel, *C How to Program (7th Ed.)*, Pearson Education , Inc., 2013.

Collaborators

- Chris Hundhausen
- Andrew O'Fallon