Control Flow

2020 Spring: Introduction to C

April 29th, 2020

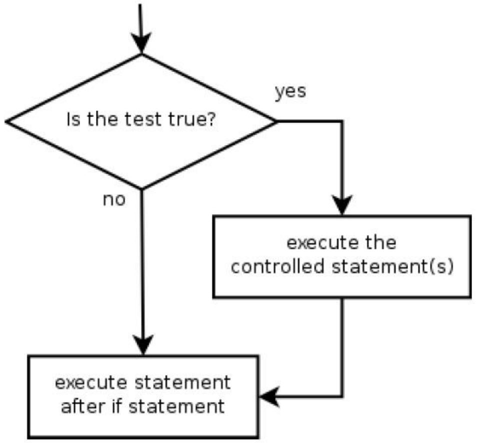
Today

- **Conditional execution**
  - if statement
  - switch statement
- **Loops**
  - while loops
  - do-while loops
  - for loops
  - break, continue

`if` Statement

- **Executes a block of statements only if a test is true**
  - Test should be evaluated to either true or false

```
if (test) {
     statement;
     ...
     statement;
}
```


Is the test true?
yes
no
execute the controlled statement(s)
execute statement after if statement

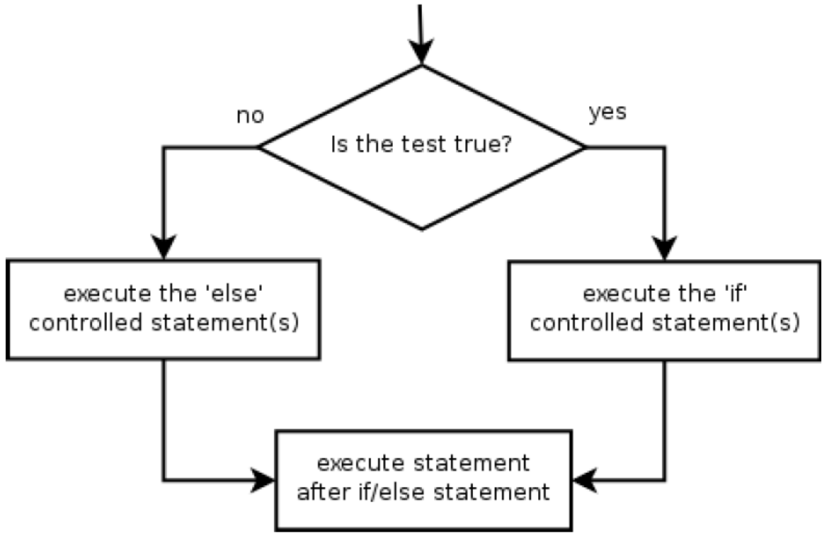
- **Example**

```
int x = 5;
if (x > 3) {
     printf("x is greater than 3");
}
```

if/else Statement

- **Executes if block if a test is true, executes else block otherwise**
  - Only one of the statements will be executed!

```
if (test) {
     statement(s);
} else {
     statement(s);
}
```


Is the test true?
no
yes
execute the 'else' controlled statement(s)
execute the 'if' controlled statement(s)
execute statement after if/else statement

- **Example**

```
int x = 5;
if (x > 3) {
     printf("x is greater than 3");
} else {
     printf("x is not greater than 3");
}
```

Misuse of `if`

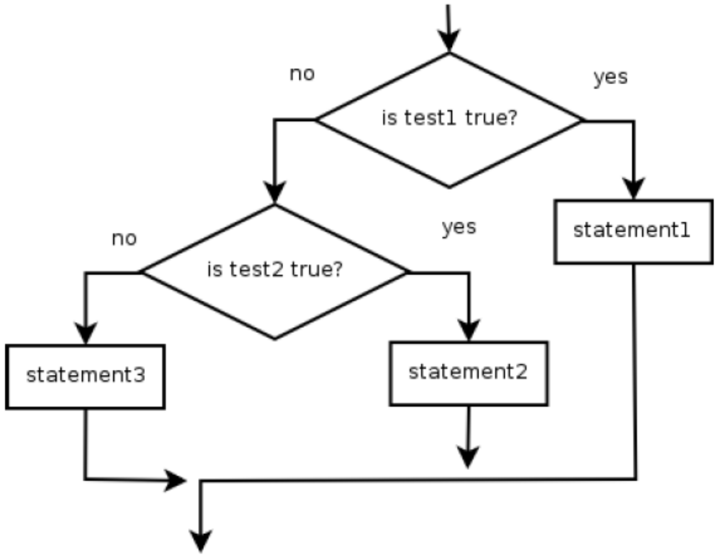
- **What's wrong with this?**

```
int percent;
printf("What percentage did you earn? ");
scanf("%d", &percent);
if (percent >= 90) {
    printf("You got an A!");
}
if (percent >= 80) {
    printf("You got a B!");
}
if (percent >= 70) {
    printf("You got a C!");
}
if (percent >= 60) {
    printf("You got a D!");
}
if (percent < 60) {
    printf("You got an F!");
}
```

else if Statement

- **Chooses between outcomes using many tests**

```
if (test) {
statement(s);
} else if (test) {
    statement(s);
} else {
    statement(s);
}
```



- **Example**

```
double y = 0;
if (y > 0) {
    printf("positive");
} else if (y < 0) {
    printf("negative");
} else {
    printf("zero");
}
```

`else if` Statement

- **If it ends with `else`, *exactly one path* must be taken**
- **If it ends with `if`, the code *might not execute* any path**

- **Example**

```
if (place == 1) {
    printf("Gold!");
} else if (place == 2) {
    printf("Silver!");
} else if (place == 3) {
    printf("Bronze!");
}
```

if/else Structures

- **Exactly 1 path**
  - Mutually exclusive

```
if (test) {
    statement(s);
} else {
    statement(s);
}
```

- **0 or 1 path**
  - Mutually exclusive

```
if (test) {
    statement(s);
} else if (test) {
    statement(s);
} else if {
    statement(s);
}
```

- **0, 1, or many paths**
  - Independent tests
  - Not exclusive

```
if (test) {
    statement(s);
}
if (test) {
    statement(s);
}
if (test) {
    statement(s);
}
```

Exercises

- #1330 두 수 비교하기
- #9498 시험 성적

Nested `if`

- **`if` can contain `if` statements**

```
int num1 = 52, num2 = 32, num3 = 1;
if (num1 > num2) {
    if (num1 > num3) {
        printf("num1 is the largest");
    }
}
```

- Try changing the condition above to a **single** `if` statement
  - *Hint: Use boolean operators!*

Dangling `else`

- **How should we interpret this code?**

```c
int num1 = 152, num2 = 173;
if (num1 > num2)
    if (num1 > 100)
        printf("num1 = %d\n", num1);
else
    if (num2 > 100)
        printf("num2 = %d\n", num2);
printf("Done.");
```

Dangling else

- **Which if statement should else be paired with?**

```
int num1 = 152, num2 = 173;
if (num1 > num2)
    if (num1 > 100)
        printf("num1 = %d\n", num1);
else
    if (num2 > 100)
        printf("num2 = %d\n", num2);
printf("Done.");
```

- **Dangling else will be paired with the *nearest* if**

Dangling `else`

- **Should be fixed this way**

```
int num1 = 152, num2 = 173;
if (num1 > num2) {
    if (num1 > 100)
        printf("num1 = %d\n", num1);
} else {
    if (num2 > 100)
        printf("num2 = %d\n", num2);
}
printf("Done.");
```

- **Use {} to explicitly mark the boundaries of `if/else` statements**
  - The code inside {} is called a ***block***

Exercises

- #10817 세 수

- #2753 윤년

switch Statement

- **expression is evaluated to an *integral value***
- **If that value equals any of val1, val2, ...**
  - The statements inside the corresponding value will be executed
  - And keeps executing the next statement until break is found
  - If corresponding value doesn't exist, statements in default is executed
  - default can be omitted

```
switch (expression) {
    case val1:
        statement(s);
        break;
    case val2:
        statement(s);
        break;
    ...
    default:
        statement(s);
        break;
}
```

`switch` Statement Example

- **What is the output?**

```
int num = 2;
switch (num) {
    case 1:
        printf("Good morning, C!\n");
        break;
    case 2:
        printf("Good afternoon, C!\n");
        break;
    case 3:
        printf("Good evening, C!\n");
        break;
    default:
        printf("Hello, C!\n");
        break;
}
```

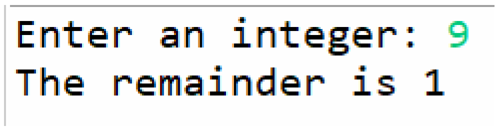
`switch` Statement Example

- **What is the output?** ***(Look out for*** **`break`** ***s)***

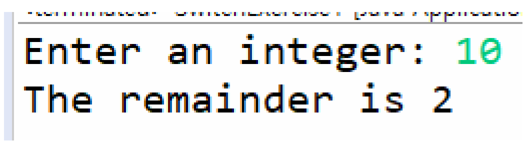
```
int num = 2;
switch (num) {
    case 1:
        printf("Good morning, C!\n");
        break;
    case 2:
        printf("Good afternoon, C!\n");
    case 3:
        printf("Good evening, C!\n");
    default:
        printf("Hello, C!\n");
        break;
}
```

Exercise

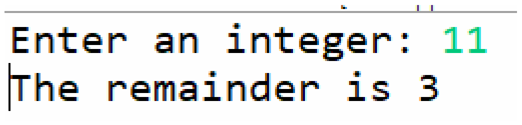
- You are given an integer. Use the switch statement to determine the remainder of that integer, when divided by 4.
- The output of your program should look like this.



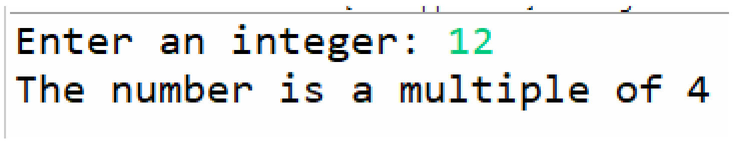
```
Enter an integer: 9
The remainder is 1
```



```
Enter an integer: 10
The remainder is 2
```



```
Enter an integer: 11
The remainder is 3
```



```
Enter an integer: 12
The number is a multiple of 4
```

Loops

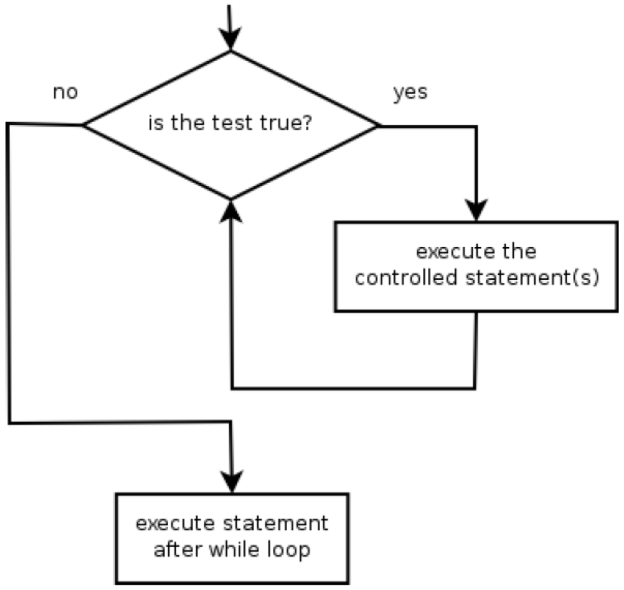
- **Definite loop:** Executes a *known number of times*
  - `for` loops are definite loops
  - Examples
    - Print "hello" 10 times
    - Find all the prime numbers up to an integer n
    - Print each odd number between 5 and 127

- **Indefinite loop:** Number of repeats is *not known in advance*
  - Examples
    - Prompt the user until they type a non-negative number
    - Print random numbers until a prime number is printed
    - Repeat until the user types "q" to quit

while Loop

- `while` loop: **Repeatedly executes its body *while* a logical test is true**

```
while (test) {
    statement(s);
}
```


no
yes
is the test true?
execute the controlled statement(s)
execute statement after while loop

- Example

```
int num = 1;                          // initialization
while (num <= 200) {                  // test
    printf("%d ", num);
    num *= 2;                         // update
}
// output: 1 2 4 8 16 32 64 128
```

Infinite loop with `while`

- **The test is checked every time!**

```
while (1) {
    printf("Stop!!!\n");
}
```

- Press **Ctrl + C** to exit out of programs that don't stop (on their own)

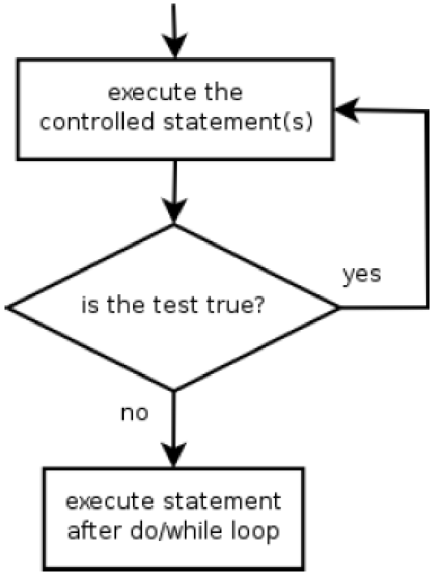
- Commonly found when ***updating procedure*** is not found

```
int num = 1;                // initialization
while (num <= 200) {        // test
    printf("%d ", num);
    // num *= 2;            // no update
}
// output: 1 2 4 8 16 32 64 128
```

do-while loop

- **Execute it body once, and execute it again *while* the test is true**
  - Performs its test at the ***end*** of each repetition
  - Guarantees that the loops body will run ***at least once***
  - Must end with a semicolon after `while`

```
do {
    statement(s);
} while (test);
```


execute the controlled statement(s)
is the test true?
yes
no
execute statement after do/while loop

- **Example**

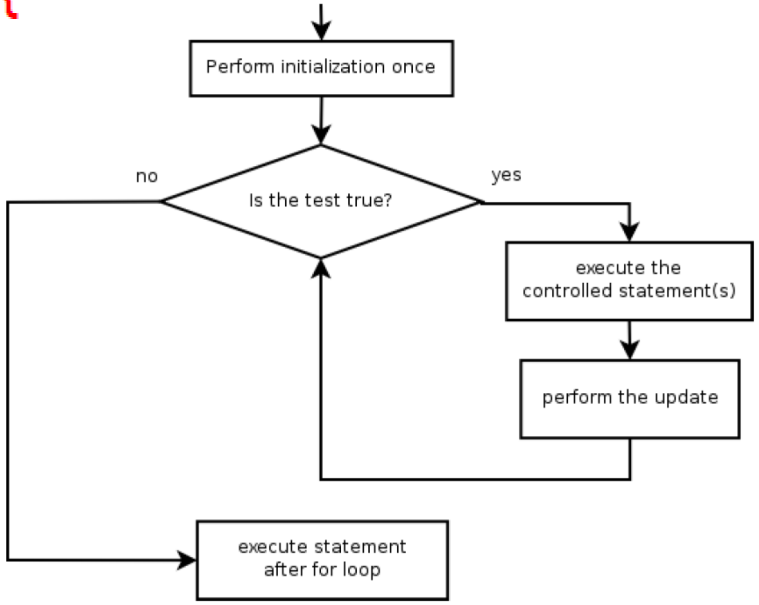
```
int x;
do {
    printf("Type in a number less than 10: ");
    scanf("%d", &x);
} while(x >= 10);
printf("OK");
```

Exercise

- **#2741 N 찍기**
- **#10950 A + B – 3**

for loop

```
for (initialization; test; update) {
    statement(s);
}
```


Perform initialization once
Is the test true?
no
yes
execute the controlled statement(s)
perform the update
execute statement after for loop

- **Perform initialization once**
- **Repeat:**
  - Check if the **test** is true. If false, stop
  - Execute the **statements**
  - Perform the **update**

for loop - Initialization

```
for (int i = 1; i <= 6; ++i) {
    printf("For Example\n");
}
```

- **Tells C what variable to use in the loop**
  - Performed **once** as the loop begins
  - The variable is called a *loop counter*
    - Can use other variable names
    - Can start at any value
    - Can initialize many variables at once

for loop - Test

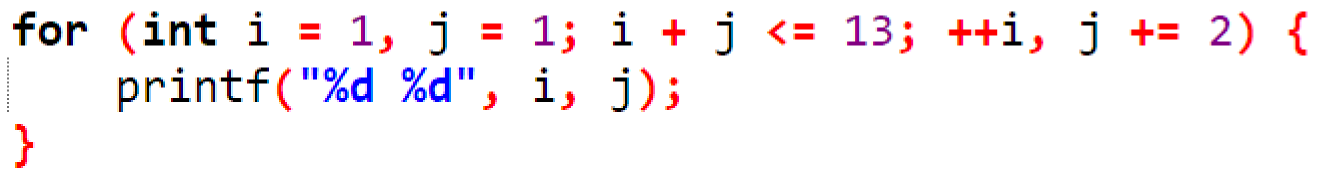
```
for (int i = 1; i <= 6; ++i) {
    printf("For Example\n");
}
```

- **Tests the expression**
  - Must be a boolean expression (evaluates to either true or false)
    - Can use complex boolean expressions
  - If true, execute the block
  - If false, stop

for loop - Update

```
for (int i = 1; i <= 6; ++i) {
    printf("For Example\n");
}
```

- **Modify the loop counter**
  - Pre/Post increment/decrement operator is used often
  - Can modify the loop counter to any value



```
for (int i = 1, j = 1; i + j <= 13; ++i, j += 2) {
    printf("%d %d", i, j);
}
```

Infinite loop with `for`

- **These are possible, and will not stop**

```
for(;;) {
    printf("Hello, C\n");
}

for(; 1; ) {
    printf("Hello, C\n");
}
```

Exercise

- #2739 구구단

- #2742 기찍 N

- #10871 X 보다 작은 수

Nested for loops

```c
for(int i = 1; i <= 5; ++i) {
    for(int j = 1; j <= 10; ++j)
        printf("*");
    printf("\n");
}
```

- **Output**

```
**********
**********
**********
**********
**********
```

- **The inner loop executes 10 times, outer loop executes 5 times**

Exercise

- **#2438 별 찍기 – 1**
- **#2439 별 찍기 – 2**

break Statement

- Used to *break out* of `for, while, do-while` loops

```
for (int i = 1; i <= 10; ++i) {
    printf("%d\n", i);
    if (i == 3)
        break;
}
printf("Done");
```

- Breaks out of loop and executes the next statement

break Statement

- In nested loops, break only breaks out of a single loop

```
for (int i = 1; i <= 3; ++i) {
    for (int j = 1; j <= 10; ++j) {
        if (j == 2)
            break;
        printf("j: %d\n", j);
    }
    printf("i: %d\n", i);
}
printf("Done");
```

- Breaks out of loop and executes the next statement

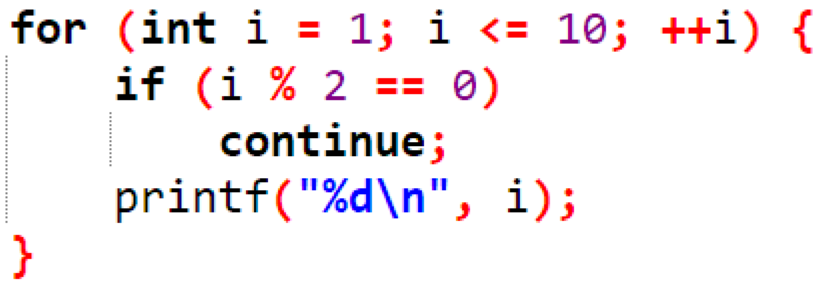
Exercise

- **#10952 A + B – 5**

continue Statement

- **Used to skip the rest of the statement and execute the next loop**

- **Example: Print odd integers from 1 to 10**



```
for (int i = 1; i <= 10; ++i) {
    if (i % 2 == 0)
        continue;
    printf("%d\n", i);
}
```

  - If `i` is even, print statement is skipped

for/while Conversion

- **for loops and while loops are interchangeable!**

```
for (initialization; test; update) {
    statement(s);
}


initialization;
while (test) {
    statement(s);
    update;
}
```