C Programming for Engineers

UNIVERSITY AT ALBANY
State University of New York

Iteration

ICEN 360– Spring 2017

Prof. Dola Saha

Application: Summing even numbers

```
1  // Fig. 4.5: fig04_05.c
2  // Summation with for.
3  #include <stdio.h>
4
5  int main(void)
6  {
7     unsigned int sum = 0; // initialize sum
8
9     for (unsigned int number = 2; number <= 100; number += 2) {
10       sum += number; // add number to sum
11    }
12
13    printf("Sum is %u\n", sum);
14 }
```

```
Sum is 2550
```

Application: Compound Interest Calculation

- Consider the following problem statement:
  - A person invests $1000.00 in a savings account yielding 5% interest. Assuming that all interest is left on deposit in the account, calculate and print the amount of money in the account at the end of each year for 10 years. Use the following formula for determining these amounts:

    $a = p(1 + r)^n$

    where

    p is the original amount invested (i.e., the principal)
    r is the annual interest rate
    n is the number of years
    a is the amount on deposit at the end of the n^{th} year.

C Code for Compound Interest Calculation

```
1  // Fig. 4.6: fig04_06.c
2  // Calculating compound interest.
3  #include <stdio.h>
4  #include <math.h>
5
6  int main(void)
7  {
8     double principal = 1000.0; // starting principal
9     double rate = .05; // annual interest rate
10
11    // output table column heads
12    printf("%4s%21s\n", "Year", "Amount on deposit");
13
14    // calculate amount on deposit for each of ten years
15    for (unsigned int year = 1; year <= 10; ++year) {
16
17       // calculate new amount for specified year
18       double amount = principal * pow(1.0 + rate, year);
19
20       // output one table row
21       printf("%4u%21.2f\n", year, amount);
22    }
23 }
```

Output

```
Year     Amount on deposit
   1              1050.00
   2              1102.50
   3              1157.63
   4              1215.51
   5              1276.28
   6              1340.10
   7              1407.10
   8              1477.46
   9              1551.33
  10              1628.89
```

Classwork Assignment

- Write a program that finds the smallest of several integers. Assume that the first value read specifies the number of values remaining. Your program should read only one value each time scanf is executed.

- A typical input sequence might be
  - 5 400 500 300 200 100
  - where 5 indicates that the subsequent five values are to be used for finding minimum.

Classwork Assignment

- Write a program that prints the following patterns separately, one below the other. Use for loops to generate the patterns. [*Hint:* The last two patterns require that each line begin with an appropriate number of blanks.]

```
(A)
*
**
***
****
*****
******
*******
********
*********
**********

(B)
**********
*********
********
*******
******
*****
****
***
**
*

(C)
**********
*********
********
*******
******
*****
****
***
**
*

(D)
         *
        **
       ***
      ****
     *****
    ******
   *******
  ********
 *********
**********
```

do ... while Iteration Statement

- Similar to the `while` statement.
- `while` (*condition*)
- The loop-continuation condition is tested at the beginning of the loop

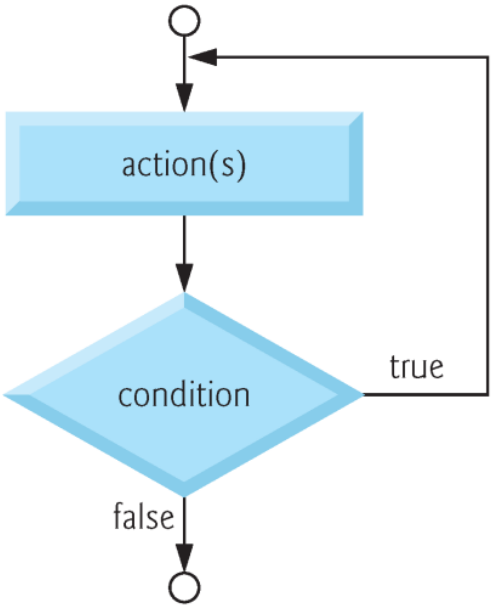
- `do`
  statement
  `while` (*condition*)`;`
- The loop-continuation condition *after* the loop body is performed.
- The loop body will be executed at least once.

Example do ... while Iteration Statement

```
1  // Fig. 4.9: fig04_09.c
2  // Using the do...while iteration statement.
3  #include <stdio.h>
4
5  int main(void)
6  {
7     unsigned int counter = 1; // initialize counter
8
9     do {
10       printf("%u  ", counter);
11    } while (++counter <= 10);
12 }
```

```
1  2  3  4  5  6  7  8  9  10
```

Flowchart do ... while Iteration Statement



`break` and `continue` Statements

- Break
  - Used inside `while, for, do...while, switch` Statements
  - When executed, program exits the statements
- Continue
  - Used in `while, for, do...while` Statements
  - When executed, the loop-continuation test is evaluated immediately *after* the `continue` statement is executed.
  - In the `for` statement, the increment expression is executed, then the loop-continuation test is evaluated.

break Statement

```
1  // Fig. 4.11: fig04_11.c
2  // Using the break statement in a for statement.
3  #include <stdio.h>
4
5  int main(void)
6  {
7     unsigned int x; // declared here so it can be used after loop
8
9     // loop 10 times
10    for (x = 1; x <= 10; ++x) {
11
12       // if x is 5, terminate loop
13       if (x == 5) {
14          break; // break loop only if x is 5
15       }
16
17       printf("%u ", x);
18    }
19
20    printf("\nBroke out of loop at x == %u\n", x);
21 }
```

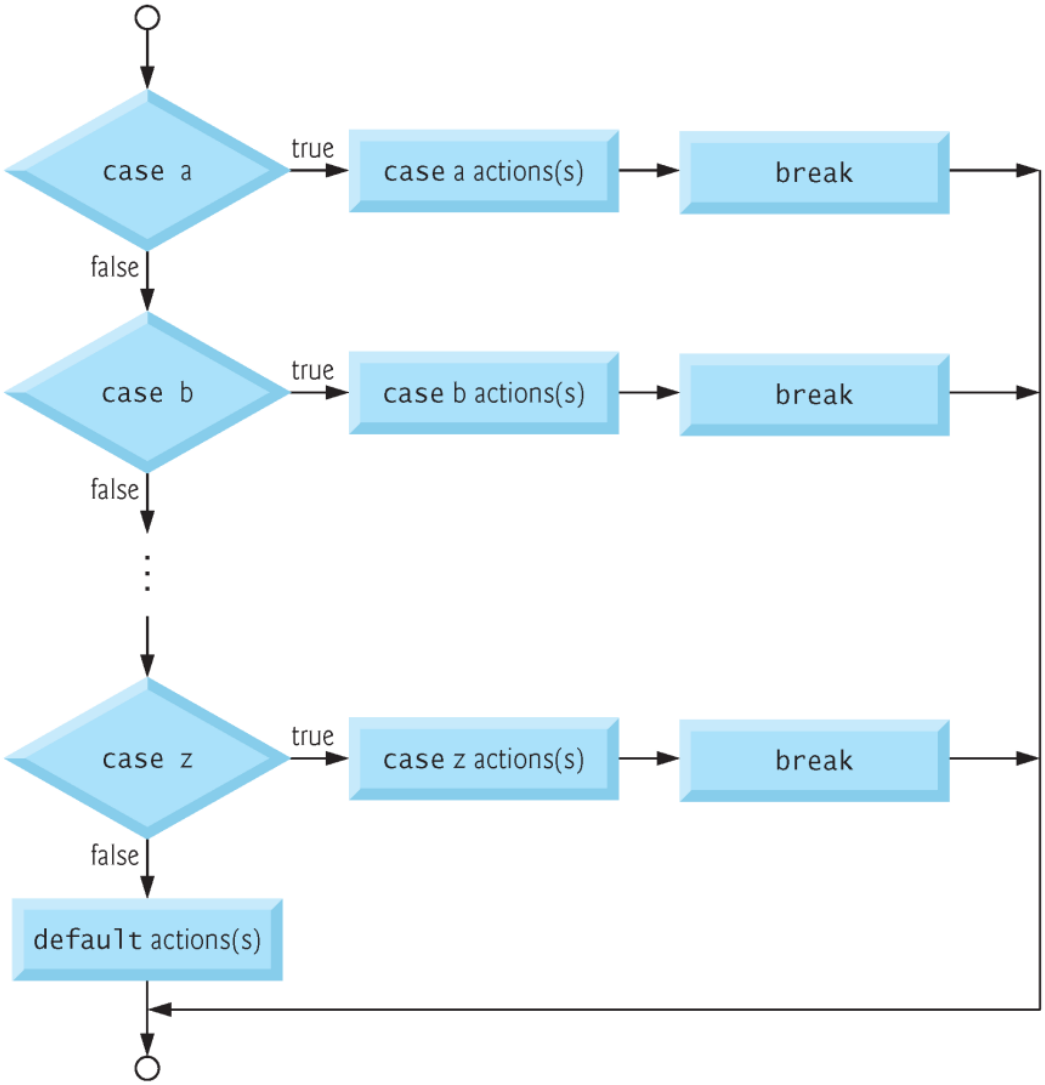
```
1 2 3 4
Broke out of loop at x == 5
```

continue Statement

```
1  // Fig. 4.12: fig04_12.c
2  // Using the continue statement in a for statement.
3  #include <stdio.h>
4
5  int main(void)
6  {
7     // loop 10 times
8     for (unsigned int x = 1; x <= 10; ++x) {
9
10       // if x is 5, continue with next iteration of loop
11       if (x == 5) {
12          continue; // skip remaining code in loop body
13       }
14
15       printf("%u ", x);
16    }
17
18    puts("\nUsed continue to skip printing the value 5");
19 }
```

```
1 2 3 4 6 7 8 9 10
Used continue to skip printing the value 5
```

Revisiting `switch` Statement



- If `break` is not used anywhere in a `switch` statement, then each time a match occurs in the statement, the statements for all the remaining `cases` will be executed—called *fallthrough*.
- If no match occurs, the `default` case is executed, and an error message is printed.

Code Snippet (1)

```
1   // Fig. 4.7: fig04_07.c
2   // Counting letter grades with switch.
3   #include <stdio.h>
4
5   int main(void)
6   {
7      unsigned int aCount = 0;
8      unsigned int bCount = 0;
9      unsigned int cCount = 0;
10     unsigned int dCount = 0;
11     unsigned int fCount = 0;
12
13     puts("Enter the letter grades.");
14     puts("Enter the EOF character to end input.");
15     int grade; // one grade
16
```

Code Snippet (2)

```
17      // loop until user types end-of-file key sequence
18      while ((grade = getchar()) != EOF) {
19
20         // determine which grade was input
21         switch (grade) { // switch nested in while
22
23            case 'A': // grade was uppercase A
24            case 'a': // or lowercase a
25               ++aCount;
26               break; // necessary to exit switch
27
28            case 'B': // grade was uppercase B
29            case 'b': // or lowercase b
30               ++bCount;
31               break;
32
33            case 'C': // grade was uppercase C
34            case 'c': // or lowercase c
35               ++cCount;
36               break;
37
```

Code Snippet (3)

```
38            case 'D': // grade was uppercase D
39            case 'd': // or lowercase d
40               ++dCount;
41               break;
42
43            case 'F': // grade was uppercase F
44            case 'f': // or lowercase f
45               ++fCount;
46               break;
47
48            case '\n': // ignore newlines,
49            case '\t': // tabs,
50            case ' ': // and spaces in input
51               break;
52
53            default: // catch all other characters
54               printf("%s", "Incorrect letter grade entered.");
55               puts(" Enter a new grade.");
56               break; // optional; will exit switch anyway
57         }
58      } // end while
59
```

Code Snippet (4) & Output

```
60      // output summary of results
61      puts("\nTotals for each letter grade are:");
62      printf("A: %u\n", aCount);
63      printf("B: %u\n", bCount);
64      printf("C: %u\n", cCount);
65      printf("D: %u\n", dCount);
66      printf("F: %u\n", fCount);
67  }
```

```
Enter the letter grades.
Enter the EOF character to end input.
a
b
c
C
A
d
f
C
E
Incorrect letter grade entered. Enter a new grade.
D
A
b
^Z ———— Not all systems display a representation of the EOF character

Totals for each letter grade are:
A: 3
B: 2
C: 3
D: 2
F: 1
```

Logical Operators

- Used to form more complex conditions by combining simple conditions.
- The logical operators are && (logical AND), || (logical OR) and ! (logical NOT also called logical negation)
- Logical AND – used to ensure that two conditions are both true before we choose a certain path of execution
- Logical OR – used to ensure that at least one condition is true before we choose a certain path of execution
- Logical NOT – used to "reverse" the meaning of a condition.

Truth Table

- Table of Logic

expression1	expression2	expression1 && expression2
0	0	0
0	nonzero	0
nonzero	0	0
nonzero	nonzero	1

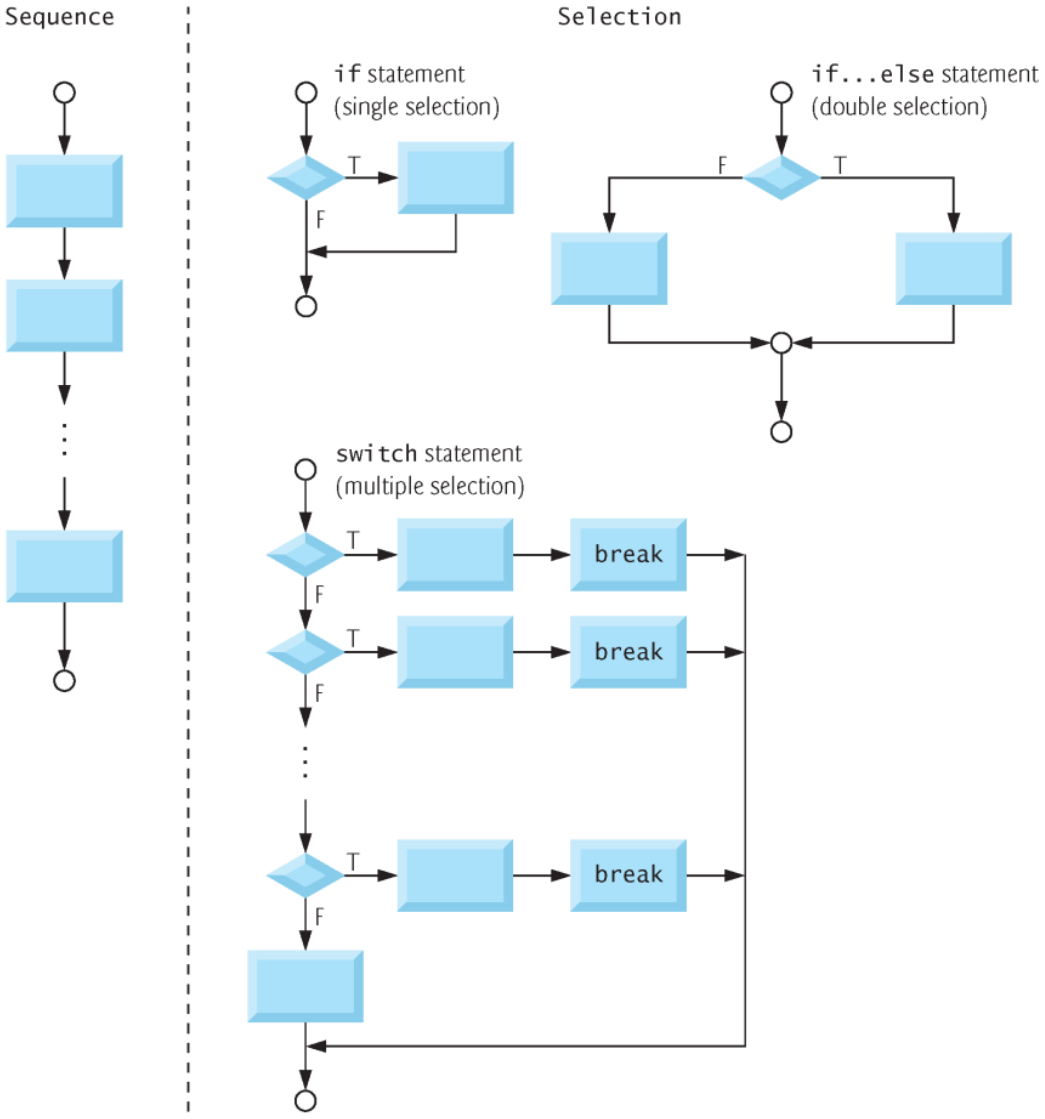
expression	!expression
0	1
nonzero	0

expression1	expression2	expression1 || expression2
0	0	0
0	nonzero	1
nonzero	0	1
nonzero	nonzero	1

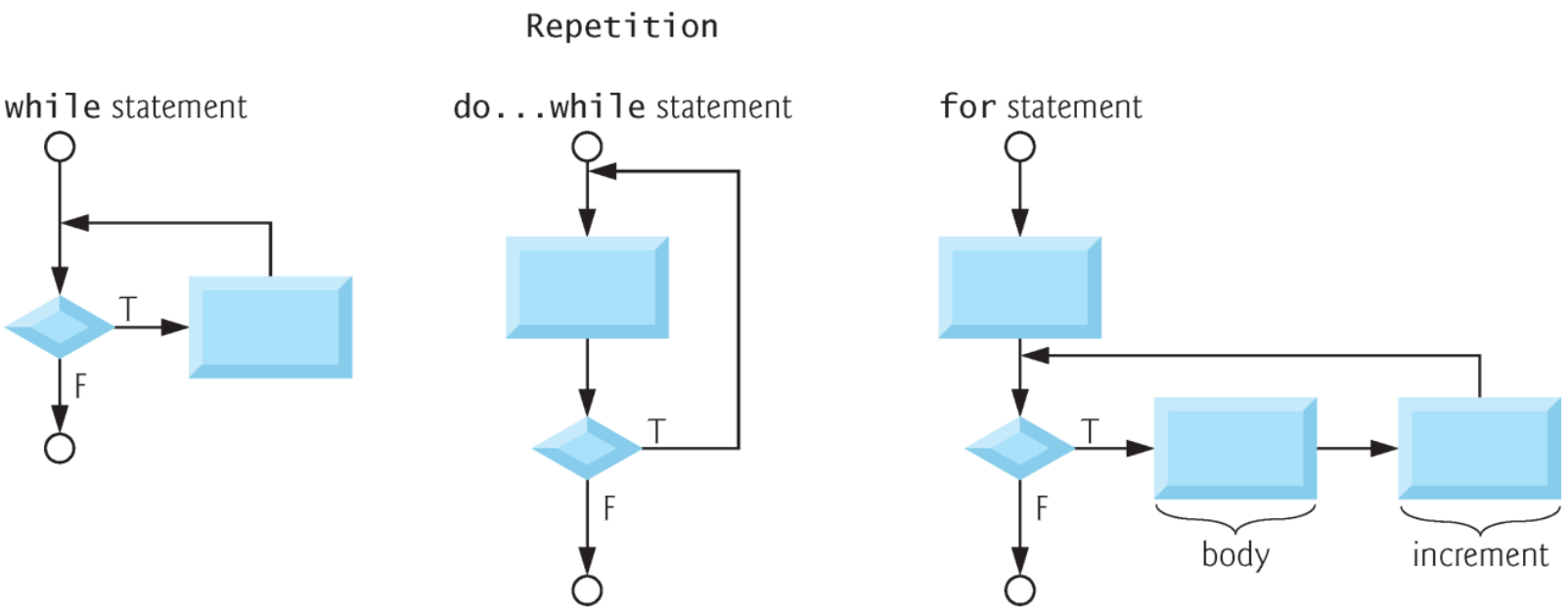
Operator Precedence

Operators	Associativity	Type
++ *(postfix)* -- *(postfix)*	right to left	postfix
+ - ! ++ *(prefix)* -- *(prefix)* (*type*)	right to left	unary
* / %	left to right	multiplicative
+ -	left to right	additive
< <= > >=	left to right	relational
== !=	left to right	equality
&&	left to right	logical AND
||	left to right	logical OR
?:	right to left	conditional
= += -= *= /= %=	right to left	assignment
,	left to right	comma

Structured Program Summary (1)



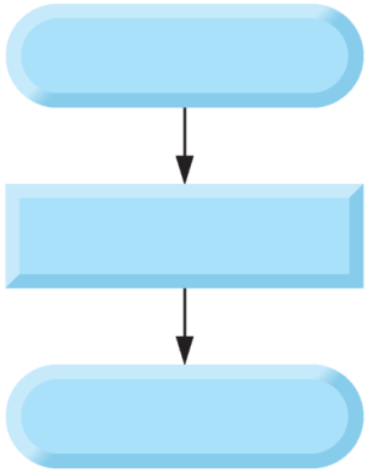
Structured Program Summary (2)



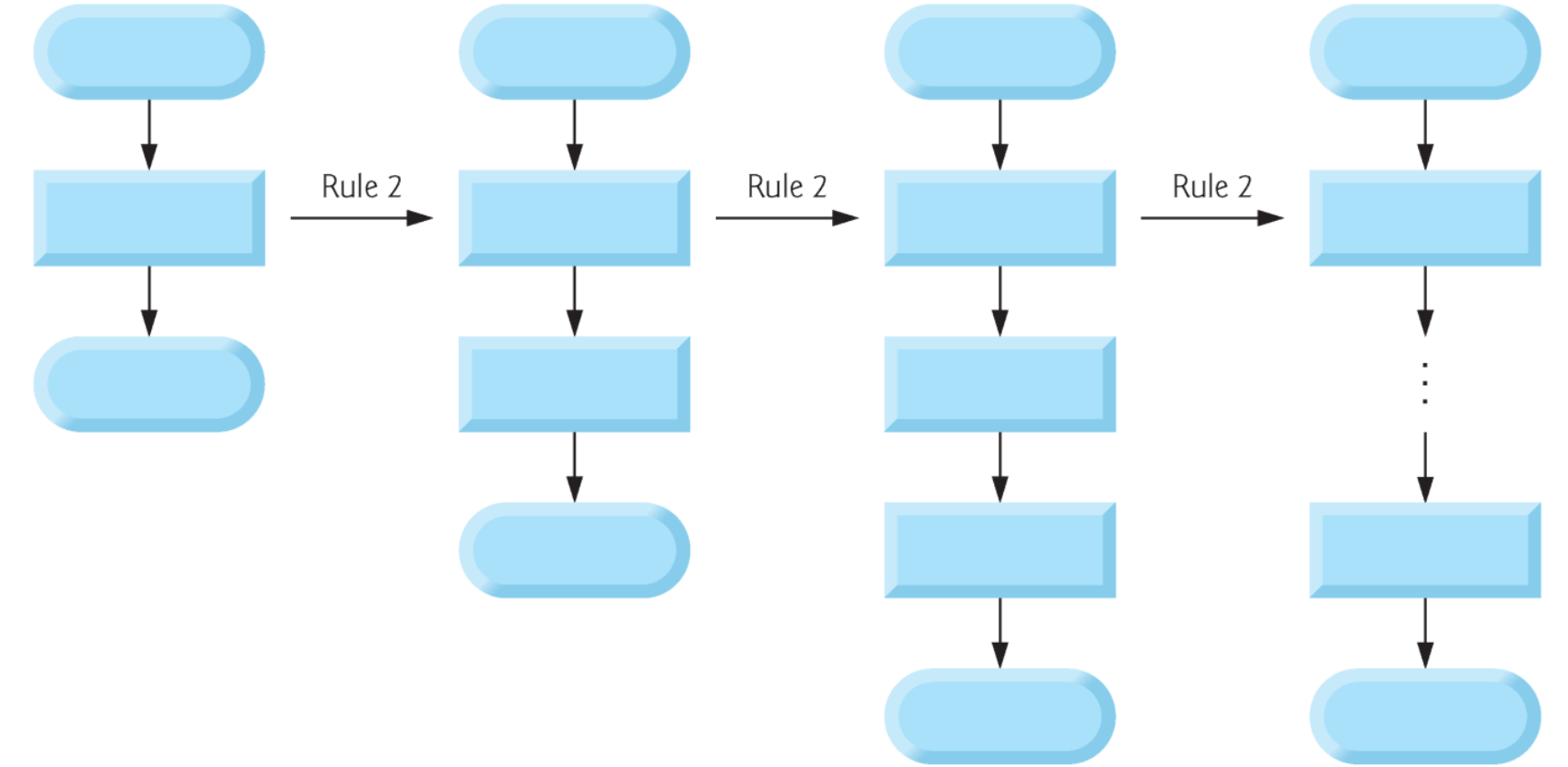
Rules for forming structured programs

- Begin with the simplest flowchart
- Stacking Rule – Any rectangle (action) can be replaced by two rectangles (actions) in sequence
- Nesting Rule – Any rectangle (action) can be replaced by any control statement
- Stacking & Nesting Rule rules may be applied in any order.

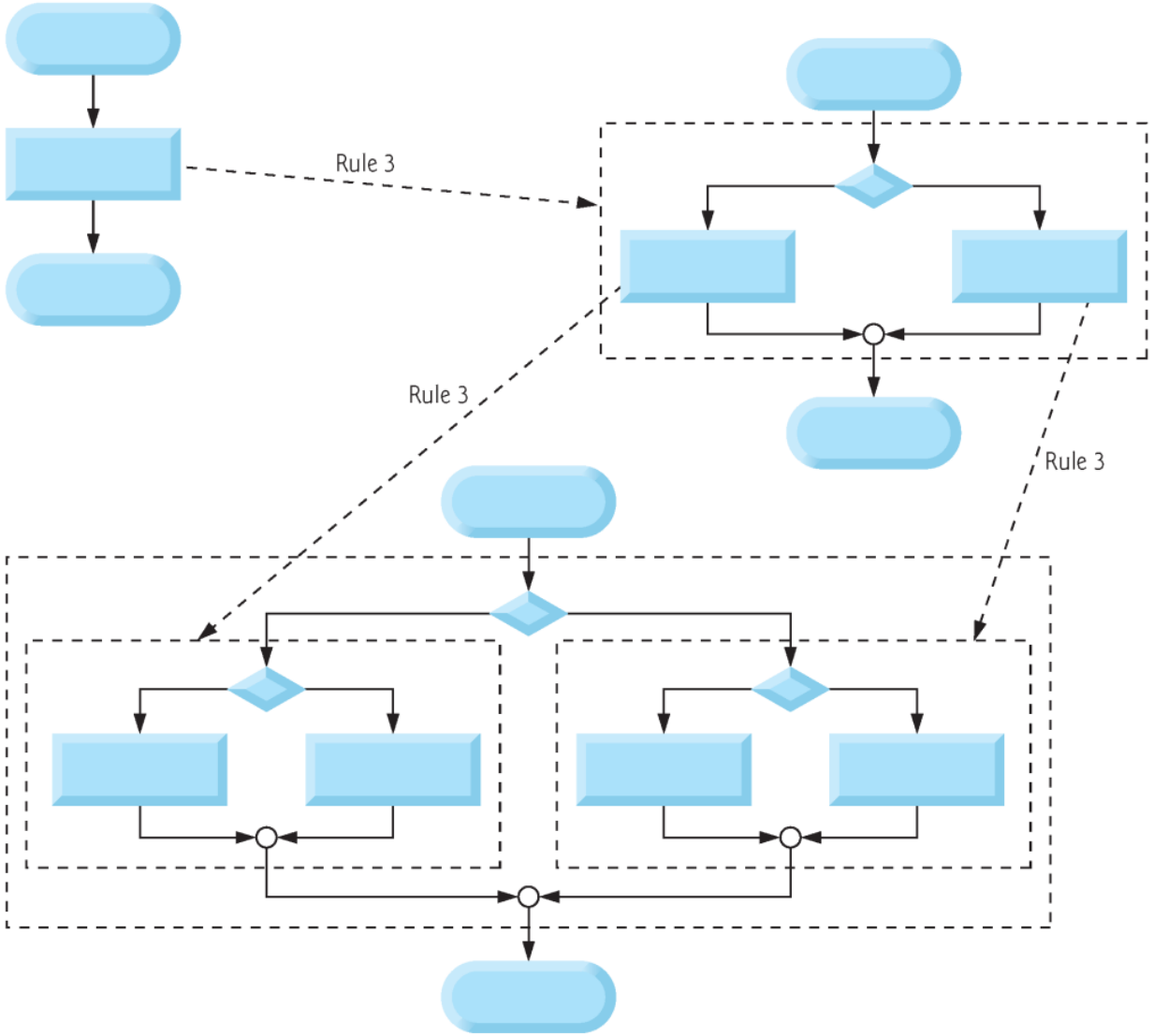
Simplest Flowchart



Stacking Rule

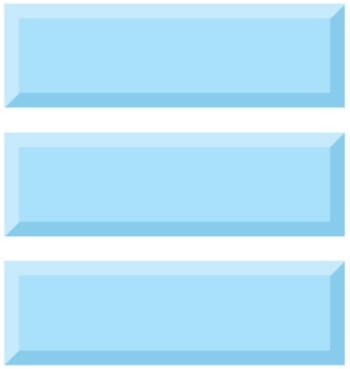


Nesting Rule

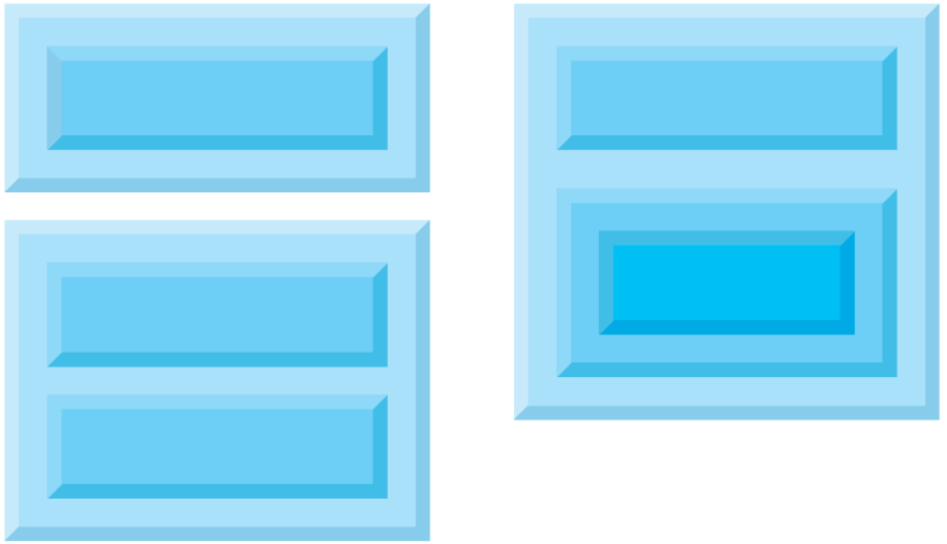


Structured Program Building Blocks

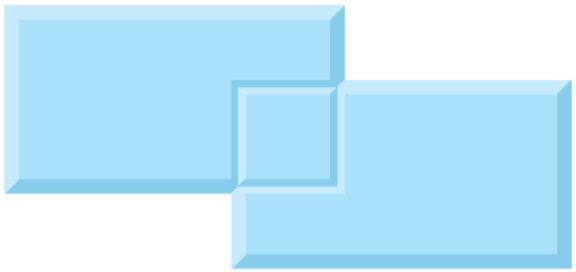
Stacked building blocks



Nested building blocks



Overlapping building blocks
(Illegal in structured programs)



Structured Programming

- Structured programming promotes simplicity.
- Bohm and Jacopini showed that only three forms of control are needed:
  - Sequence
  - Selection
  - Iteration

Structured Programming Options

- Sequence is straightforward.
- Selection is implemented in one of three ways:
  - `if` statement (single selection)
  - `if`…`else` statement (double selection)
  - `switch` statement (multiple selection)
- Iteration is implemented in one of three ways:
  - `while` statement
  - `do`…`while` statement
  - `for` statement