Week 3

Control Structures

Akbayan Bekarystankyzy
akbayan.b@gmail.com

2.15 Essentials of Counter-Controlled Repetition

- Counter-controlled repetition requires
  - Name of control variable/loop counter
  - Initial value of control variable
  - Condition to test for final value
  - Increment/decrement to modify control variable when looping

```cpp
// Fig. 2.16: fig02_16.cpp
// Counter-controlled repetition.
#include <iostream>

using std::cout;
using std::endl;

// function main begins program execution
int main()
{
   int counter = 1;              // initialization

   while ( counter <= 10 ) {    // repetition condition
      cout << counter << endl;  // display counter
      ++counter;                // increment

   } // end while

   return 0;   // indicate successful termination

} // end function main
```

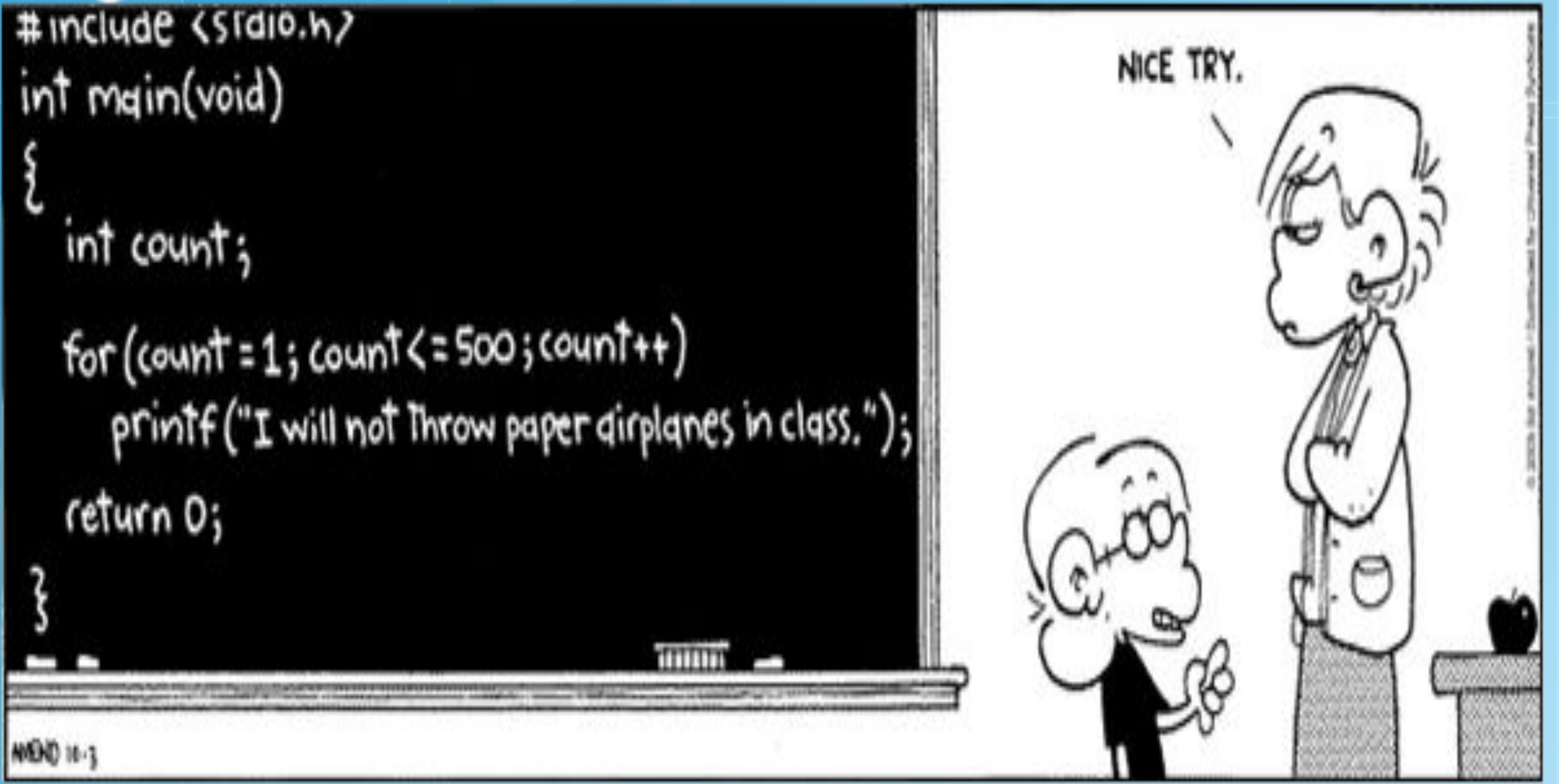
```
1
2
3
4
5
6
7
8
9
10
```

2.15 Essentials of Counter-Controlled Repetition

- The declaration

```
        int counter = 1;
```

  - Names **counter**
  - Declares **counter** to be an integer
  - Reserves space for **counter** in memory
  - Sets **counter** to an initial value of **1**



```
#include <stdio.h>
int main(void)
{
   int count;

   for(count=1; count<=500; count++)
      printf("I will not Throw paper airplanes in class.");

   return 0;
}
```

2.16 for Repetition Structure

- General format when using **for** loops

```
for ( initialization; LoopContinuationTest; increment )
    statement
```

- Example

```
for( int counter = 1; counter <= 10; counter++ )
   cout << counter << endl;
```

  - Prints integers from one to ten

No semicolon after last statement

```cpp
// Fig. 2.17: fig02_17.cpp
// Counter-controlled repetition with the for structure.
#include <iostream>

using std::cout;
using std::endl;

// function main begins program execution
int main()
{
   // Initialization, repetition condition and incrementing
   // are all included in the for structure header.

   for ( int counter = 1; counter <= 10; counter++ )
      cout << counter << endl;

   return 0;   // indicate successful termination

} // end function main
```

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

2.16 for Repetition Structure

- **for** loops can usually be rewritten as **while** loops

```
initialization;
while ( loopContinuationTest){
   statement
   increment;
}
```

- Initialization and increment
  - For multiple variables, use comma-separated lists

```
for (int i = 0, j = 0;  j + i <= 10; j++,
  i++)
   cout << j + i << endl;
```

```cpp
// Fig. 2.20: fig02_20.cpp
// Summation with for.
#include <iostream>

using std::cout;
using std::endl;

// function main begins program execution
int main()
{
   int sum = 0;                         // initialize sum

   // sum even integers from 2 through 100
   for ( int number = 2; number <= 100; number += 2 )
      sum += number;                    // add number to sum

   cout << "Sum is " << sum << endl;  // output sum
   return 0;                            // successful termination

} // end function main
```

```
Sum is 2550
```

2.17 Examples Using the for Structure

- Program to calculate compound interest
- *A person invests $1000.00 in a savings account yielding 5 percent 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:*

  $a = p(1+r)^n$
- *p* is the original amount invested (i.e., the principal),
- *r* is the annual interest rate,
- *n* is the number of years and
- *a* is the amount on deposit at the end of the *n*-th year

```
// Fig. 2.21: fig02_21.cpp
// Calculating compound interest.
#include <iostream>

using std::cout;
using std::endl;
using std::ios;
using std::fixed;

#include <iomanip>

using std::setw;
using std::setprecision;

#include <cmath>  // enables program to use function pow

// function main begins program execution
int main()
{
   double amount;              // amount on deposit
   double principal = 1000.0;  // starting principal
   double rate = .05;          // interest rate

```

`<cmath>` header needed for the `pow` function (program will not compile without it).

```
   // output table column heads
   cout << "Year" << setw( 21 ) << "Amount on deposit" << endl;

   // set floating-point number format
   cout << fixed << setprecision( 2 );

   // calculate amount on deposit for each of ten years
   for ( int year = 1; year <= 10; year++ ) {

      // calculate new amount for specified year
      amount = principal * pow( 1.0 + rate, year );

      // output one table row
      cout << setw( 4 ) << year
           << setw( 21 ) << amount << endl;

   } // end for

   return 0;   // indicate successful termination

} // end function main
```

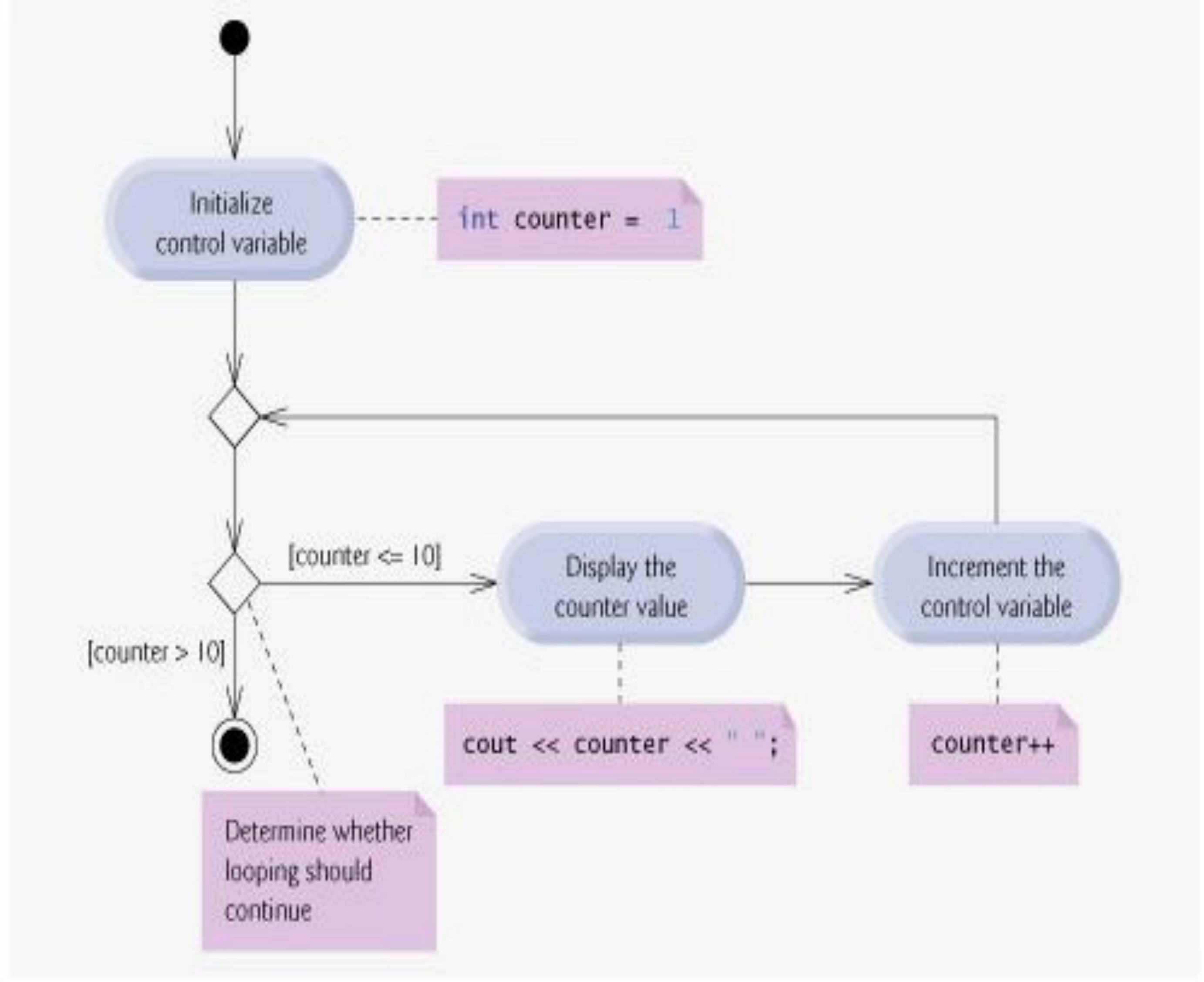
Sets the field width to at least 21 characters. If output less than 21, it is right-justified.

`pow(x,y)` = x raised to the yth power.

```
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
```

Numbers are right-justified due to setw statements (at positions 4 and 21).

2.17.1 FOR REPETITION STRUCTURE FLOWCHART


Initialize control variable
Display the counter value
Increment the control variable
[counter <= 10]
[counter > 10]
Determine whether looping should continue

```
int counter = 1
```

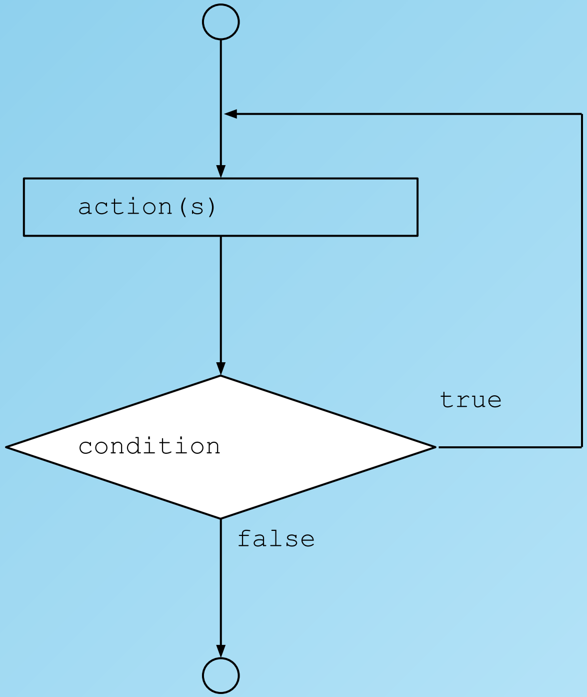
```
cout << counter << " ";
```

```
counter++
```

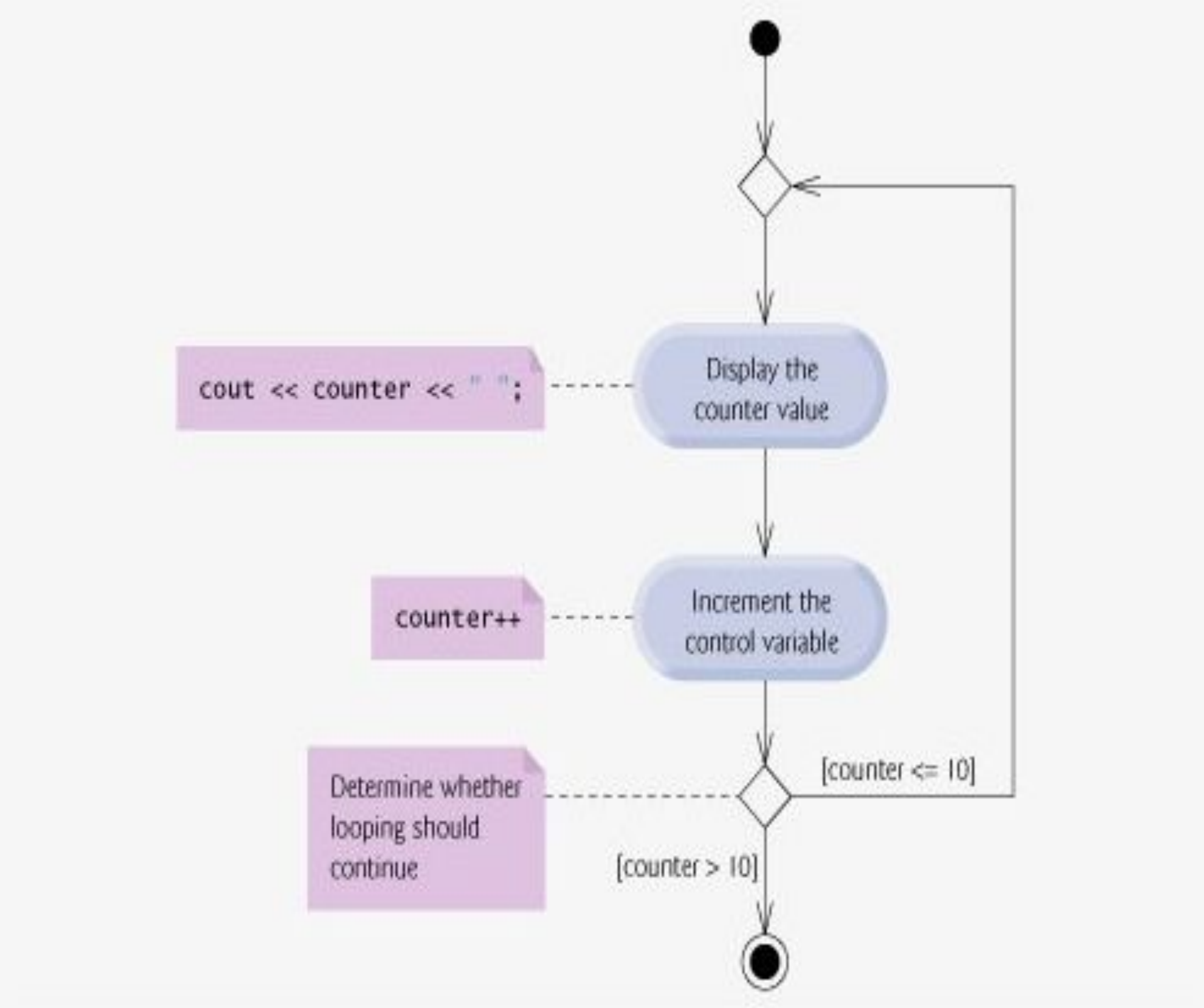
2.19 do/while Repetition Structure

- Similar to **`while`** structure
  - Makes loop continuation test at end, not beginning
  - Loop body executes at least once
- Format

```
do {
    statement
} while ( condition );
```



2.19 do/while Repetition Structure



```
cout << counter << " ";
```

```
counter++
```

Display the counter value
Increment the control variable
Determine whether looping should continue
[counter <= 10]
[counter > 10]

```
// Fig. 2.24: fig02_24.cpp
// Using the do/while repetition structure.
#include <iostream>

using std::cout;
using std::endl;

// function main begins program execution
int main()
{
   int counter = 1;              // initialize counter

   do {
      cout << counter << "  ";   // display counter
   } while ( ++counter <= 10 );  // end do/while

   cout << endl;

   return 0;   // indicate successful termination

} // end function main
```

Notice the preincrement in loop-continuation test.

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

2.20 break and continue Statements

- **`break`** statement
  - Immediate exit from **`while`**, **`for`**, **`do/while`**, **`switch`**
  - Program continues with first statement after structure
- Common uses
  - Escape early from a loop
  - Skip the remainder of **`switch`**

```cpp
// Fig. 2.26: fig02_26.cpp
// Using the break statement in a for structure.
#include <iostream>

using std::cout;
using std::endl;

// function main begins program execution
int main()
{

   int x;  // x declared here so it can be used after the loop

   // loop 10 times
   for ( x = 1; x <= 10; x++ ) {

      // if x is 5, terminate loop
      if ( x == 5 )
         break;         // break loop only if x is 5

      cout << x << " ";   // display value of x

   } // end for

   cout << "\nBroke out of loop when x became " << x << endl;
```

Exits `for` structure when `break` executed.

```

   return 0; // indicate successful termination

} // end function main
```

```
1 2 3 4
Broke out of loop when x became 5
```

2.20 break and continue Statements

- **`continue`** statement
  - Used in **`while`**, **`for`**, **`do/while`**
  - Skips remainder of loop body
  - Proceeds with next iteration of loop
- **`while`** and **`do/while`** structure
  - Loop-continuation test evaluated immediately after the **`continue`** statement
- **`for`** structure
  - Increment expression executed
  - Next, loop-continuation test evaluated

```cpp
// Fig. 2.27: fig02_27.cpp
// Using the continue statement in a for structure.
#include <iostream>

using std::cout;
using std::endl;

// function main begins program execution
int main()
{
   // loop 10 times
   for ( int x = 1; x <= 10; x++ ) {

      // if x is 5, continue with next iteration of loop
      if ( x == 5 )
         continue;       // skip remaining code in loop body

      cout << x << " ";   // display value of x

   } // end for structure

   cout << "\nUsed continue to skip printing the value 5"
        << endl;

   return 0;              // indicate successful termination
```

Skips to next iteration of the loop.

```

} // end function main
```

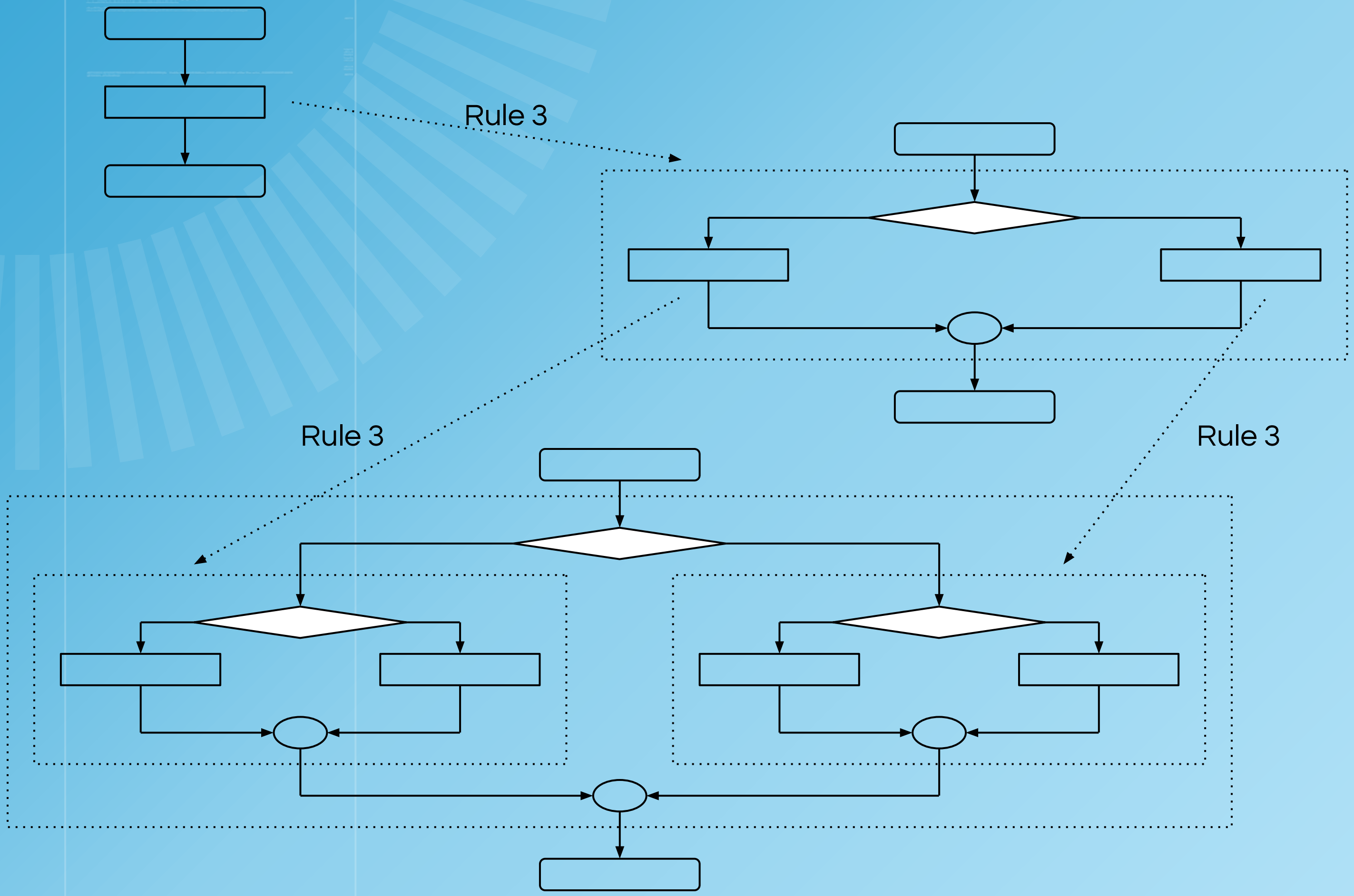
```
1 2 3 4 6 7 8 9 10
Used continue to skip printing the value 5
```

2.21 Structured-Programming Summary

- Structured programming
  - Programs easier to understand, test, debug and modify
- Rules for structured programming
  - Only use single-entry/single-exit control structures
  - Rules
    1) Begin with the “simplest flowchart”
    2) Any rectangle (action) can be replaced by two rectangles (actions) in sequence
    3) Any rectangle (action) can be replaced by any control structure (sequence, if, if/else, switch, while, do/while or for)
    4) Rules 2 and 3 can be applied in any order and multiple times

2.21 Structured-Programming Summary

Representation of Rule 3 (replacing any rectangle with a control structure)



2.21 Structured-Programming Summary

- All programs broken down into
  - Sequence
  - Selection
    - **`if`**, **`if/else`**, or **`switch`**
    - Any selection can be rewritten as an **`if`** statement
  - Repetition
    - **`while`**, **`do/while`** or **`for`**
    - Any repetition structure can be rewritten as a **`while`** statement

Readings:

- **C++ How to Program,** By H. M. Deitel
  - Chapter 5. Control Statements: Part 2

Thanks for your attention!