



CONTROL FLOW

Sevil ŞEN

Hacettepe University

November 2010

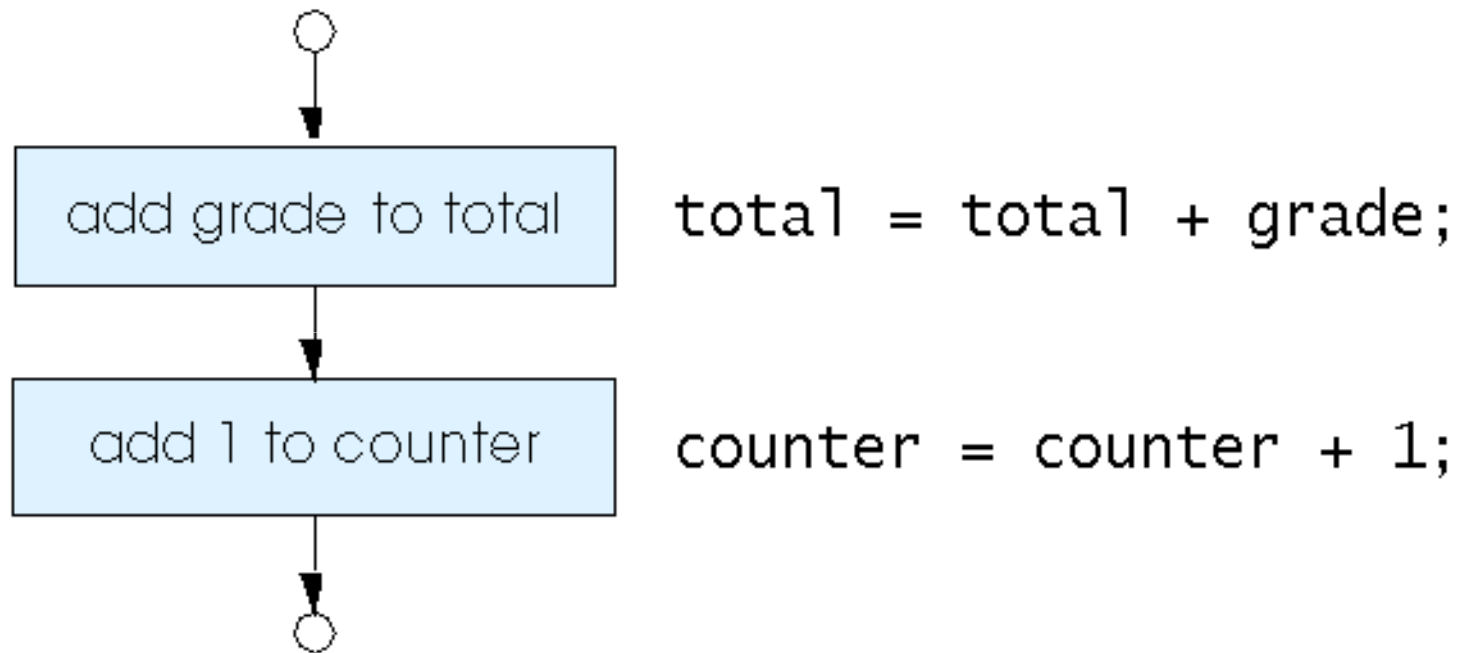
Content

- Sequence structures: Programs executed sequentially by default
- Selection structures: `if`, `if...else`, and `switch`
- Repetition structures: `while`, `do...while` and `for`

In this lecture, you will learn

- To be able to use the **if** selection statement and **if...else** selection statement to select actions.
- To be able to use the **while** and **for** repetition statements to write loops (the repetition of steps in a program).

Sequential Structure



Flowcharting C's sequence structure

Relational Operators

- The relational operators are `<`, `>`, `<=`, and `>=`.
- Take **2** expressions as operands
- Yield either the `int` value **0** (false) or the `int` value **1** (true).

Valid

`a < 3`
`a > b`

Invalid

`a =< b`
`a < = b`

Examples: assume `a = 1`, `b=2`.

Expression

`a <= b`

`a < b-5`

`a + 10 / b <= -3 + 8`

Value

1

0

0

Equality Operators

- The equality operators are `==` and `!=`.
- Yield either the `int` value **0** or the `int` value **1**.

Valid

`x != -2.77`

`x + 2.0 != 3.3/z`

`ch == '*'`

Invalid

`x = = y-1`

`x =! 44`

`ch = '*'`

Examples: assume `a=1`, `b=2`, `ch = 'A'`

Expression

`a == b`

`a != b`

`ch < 'B'`

`a+b == -2 * 3`

Value

0

1

1

0

Equality Operators

- Note carefully that the two expressions

`a == b` and `a = b`

are visually similar.

- The expression

`a == b` is a test for equality.

`a = b` is an assignment expression.

Logical Operators

- The logical operators are **&&**, **||**, and **!**.
- Expressions connected by **&&** or **||** are evaluated left to right.
- Logical negation: **!**

Value of *expression*

zero

nonzero

!*expression*

1

0

Expression

Value

!5

0

!!5

1

!(6 < 7)

0

!6 < 7

1

!(3-4)

0

Logical Operators

a	b	a&&b	a b
zero	zero	0	0
zero	nonzero	0	1
nonzero	zero	0	1
nonzero	nonzero	1	1

Examples

- Given declarations:

```
int a = 3, b = 3, c = 3;  
double x = 0.0, y = 2.5;  
char ch = 'g'
```

Expression

Value

<code>!(a < b) && c</code>	1
<code>ch >= 'a' && ch <= 'z'</code>	1
<code>x a && b - 3</code>	0
<code>a < b && x < y</code>	0
<code>a < b x < y</code>	1

- The precedence of `&&` is higher than `||`, but both operators are of lower precedence than all unary, arithmetic and relational operators. Their associativity is left to right.

Short-Circuit Evaluation

- For the expressions that contain the operands of `&&` and `||`, the expression process stops as soon as the outcome true or false is known.
- Suppose `expr1` is 0.
`expr1 && expr2 = 0` (expr2 will not be evaluated.)
- Suppose `expr1` is nonzero.
`expr1 || expr2 = 1` (expr2 will not be evaluated.)

Operator Precedence

Operators						Associativity	Type
++	--	+	-	!	(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

Fig. 4.16 Operator precedence and associativity.

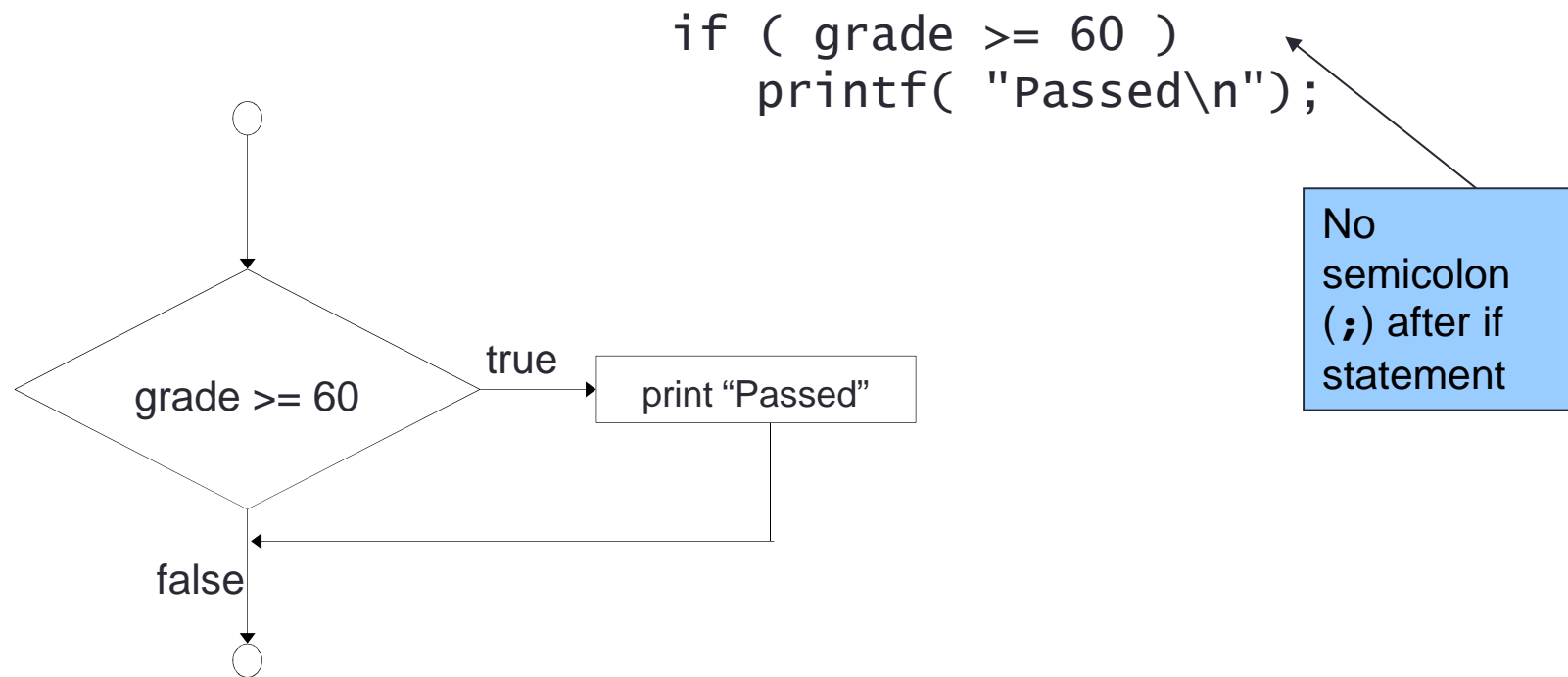


SELECTION STRUCTURE

The `if` Selection Statement

- Selection structure:
 - Used to choose among alternative courses of action
 - Pseudocode:
If student's grade is greater than or equal to 60
Print "Passed"
- If condition `true`
 - Print statement executed and program goes on to next statement
 - If `false`, print statement is ignored and the program goes onto the next statement
 - Indenting makes programs easier to read
 - C ignores whitespace characters

The if Selection Statement



Write a program that determines if a number entered by the user is even

```
/* Determines if a number is even */
#include <stdio.h>

int main(void)
{
    int value;

    printf("Enter a number.\n");
    scanf("%d",&value);
    if(value % 2 == 0)
        printf("\n%d is an even number.\n",value);
    return 0;
}
```

Write a program that prints the maximum of two numbers entered by the user

```
#include <stdio.h>

int main ( )
{
    int value1, value2, max=0;

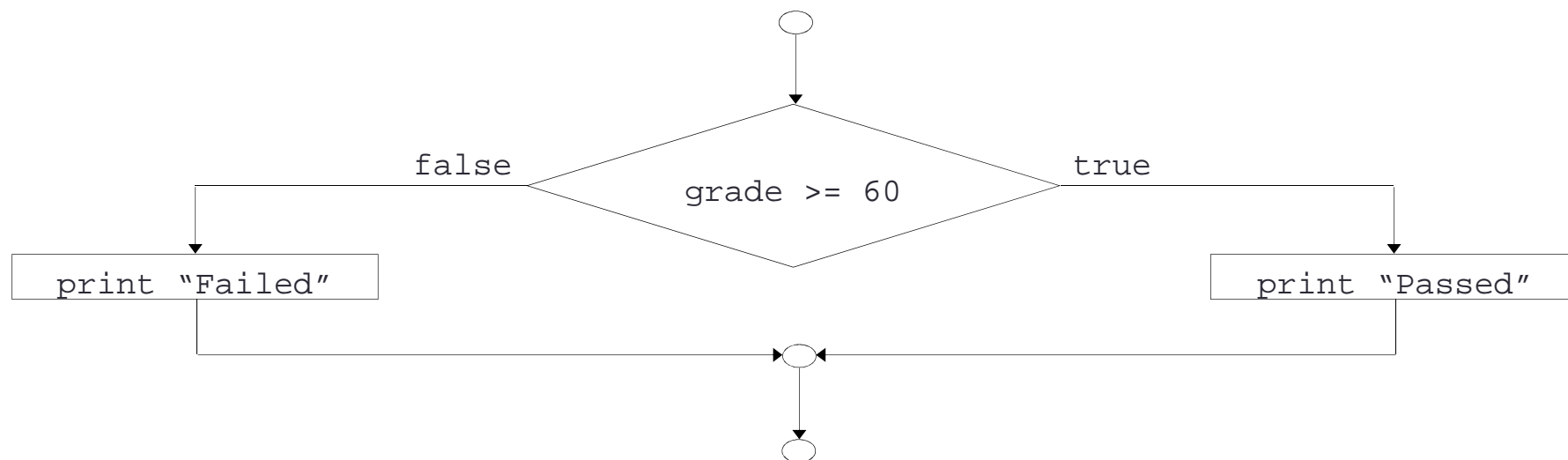
    printf("Enter two values:\n");
    scanf("%d %d", &value1, &value2);
    if(value1 > value2)
        max = value1;
    if(value1 <= value2)
        max = value2;
    printf("%d\n", max);
    return 0;
}
```


The `if...else` Selection Statement

- `if`
 - Only performs an action if the condition is `true`
- `if...else`
 - Specifies an action to be performed both when the condition is `true` and when it is `false`
- Psuedocode:
 - If student's grade is greater than or equal to 60*
Print "Passed"
 - else*
Print "Failed"
- Note spacing/indentation conventions

The `if...else` Selection Statement

- Flow chart of the `if...else` selection statement



The `if...else` Selection Statement

- Compound statement:
 - Set of statements within a pair of braces

Example:

```
if ( grade >= 60 )
    printf( "Passed.\n" );
else {
    printf( "Failed.\n" );
    printf( "You must take this course again.\n" );
}
```

- Without the braces, the statement

```
printf( "You must take this course again.\n" );
```


would be executed automatically

Write a program that prints the maximum of two numbers entered by the user

```
#include <stdio.h>

int main ( )
{
    int value1, value2, max=0;

    printf("Enter two values:\n");
    scanf("%d %d", &value1, &value2);
    if (value1 > value2)
        max = value1;
    else
        max = value2;
    printf("%d\n", max);
    return 0;
}
```

Examples

- Dangling else: an `else` attaches to the nearest `if` .

```
if(a == 10)
    if(b==20)
        printf( "***\n" );
    else
        printf( "###\n" );
```

The `if...else` Selection Statement

- Ternary conditional operator (`?:`)
 - Takes three arguments (condition, value if true, value if false)
 - Our pseudocode could be written:
`printf("%s\n", grade >= 60 ? "Passed" : "Failed");`
 - Or it could have been written:
`grade >= 60 ? printf("Passed\n") : printf("Failed\n");`

Write a program that prints the minimum of three numbers entered by the user

```
// Find the minimum of three values.
#include <stdio.h>

int main()
{
    int a, b, c, min;

    printf("Enter three numbers:");
    scanf("%d%d%d", &a,&b,&c);

    if(a < b)
        min = a;
    else
        min = b;
    if(c < min)
        min = c;

    printf("The minimum value is %d\n", min);
    return 0;
}
```

Nested `if/else` structures

- Nested `if...else` statements
 - Test for multiple cases by placing `if...else` selection statements inside `if...else` selection statement
 - Once condition is met, rest of statements skipped
 - Deep indentation usually not used in practice

The if...else Selection Statement

- Pseudocode for a nested if...else statement

If student's grade is greater than or equal to 90

Print "A"

else

If student's grade is greater than or equal to 80

Print "B"

else

If student's grade is greater than or equal to 70

Print "C"

else

If student's grade is greater than or equal to 60

Print "D"

else

Print "F"

Nested if/else structures

- Its general form is:

```
if (expr1)
    statement1
else if(expr2)
    statement2
else if(expr3)
    statement3
.....
else if(exprN)
    statementN
else
    default statement
next statement
```

Nested if's

```
if(grade >= 90)
    printf("A");
else if (grade >= 80)
    printf("B");
else if (grade >= 70)
    printf("C");
else if (grade >= 60)
    printf("D");
else
    printf("F");
```

The `switch` Multiple-Selection Structure

- **switch**

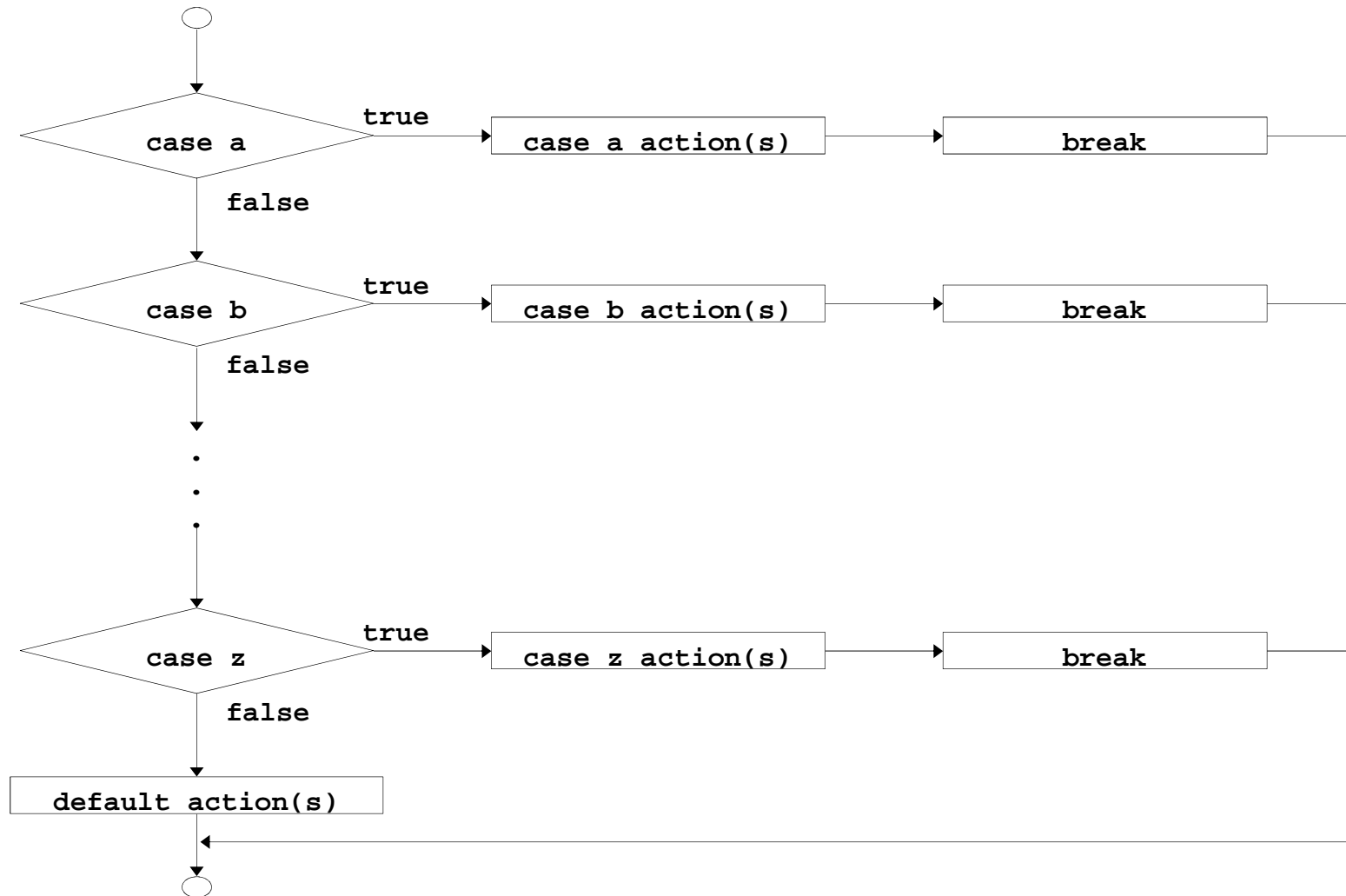
- Useful when a variable or expression is tested for all the values it can assume and different actions are taken

- Series of case labels and an optional default case

```
switch ( a_variable ){  
    case value1:  
        actions  
    case value2 :  
        actions  
    ...  
    default:  
        actions  
}
```

- `break`; exits from structure

The switch Multiple-Selection Structure



A program to count letter (upper case) grades

```
/*Counting letter grades */
```

```
char grade;
```

```
int aCount = 0, bCount = 0, cCount = 0,  
    dCount = 0, fCount = 0;
```

```
printf("Enter the letter grade.\n" );  
scanf("%c",&grade);
```

```
switch ( grade ) {
```

```
    case 'A': ++aCount;  
              break;
```

```
    case 'B': ++bCount;  
              break;
```

```
    case 'C': ++cCount;  
              break;
```

```
    case 'D': ++dCount;  
              break;
```

```
    case 'F': ++fCount;  
              break;
```

```
    default: /* catch all other characters */  
              printf( "Incorrect letter grade entered." );  
              printf( " Enter a new grade.\n" );  
              break;
```

```
}
```

A program to count letter (upper/lower case) grades

```
/*Counting letter grades */
```

```
char grade;
```

```
int aCount = 0, bCount = 0, cCount = 0,  
    dCount = 0, fCount = 0;
```

```
printf("Enter the letter grade.\n" );  
scanf("%c",&grade);
```

```
switch ( grade ) {
```

```
    case 'A':
```

```
    case 'a': ++aCount;  
              break;
```

```
    case 'B':
```

```
    case 'b': ++bCount;  
              break;
```

```
    case 'C':
```

```
    case 'c': ++cCount;  
              break;
```

```
    case 'D':
```

```
    case 'd': ++dCount;  
              break;
```

```
    case 'F':
```

```
    case 'f': ++fCount;  
              break;
```

```
    default:
```


```
        /* catch all other characters */
```

```
        printf( "Incorrect letter grade entered." );
```

```
        printf( " Enter a new grade.\n" );
```

```
        break;
```

```
}
```



Write a program that finds how many days are in the month/year entered by the user.

April, June, September, November: 30 days

February : 28 days (if it is a leap year, then 29 days)

Other : 31 days

Leap Year:

- most years that are evenly divisible by 4 are leap years
- years that are evenly divisible by 100 are *not* leap years, unless they are also evenly divisible by 400.


```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int month, year, days, leapyear;
```

```
    printf("Enter a month and a year:");
```

```
    scanf("%d%d", &month, &year);
```

```
    if(((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0))
```

```
        leapyear = 1;
```

```
    else
```

```
        leapyear = 0;
```

```
    switch(month){
```

```
        case 9 :
```

```
        case 4 :
```

```
        case 6 :
```

```
        case 11: days=30;
```

```
            break;
```

```
        case 2 : days = (leapyear == 1)? 29: 28;
```

```
            break;
```

```
        default :
```

```
            days = 31;
```

```
    }
```

```
    printf("There are %d days in that month in that year.\n", days);
```

```
    return 0;
```

```
}
```

Exercises

1. Describe what the following conditional expression means:

$(x \neq 4) \parallel (x \neq 17)$

2. Write a conditional expression that is true
 - i. if exactly one of a and b is true.
 - ii. if both a and b are true, or both are false.

Exercises

3. Write a C statement that
 - i. classifies a given character as an uppercase letter, a lowercase letter or a special character.
 - ii. If the values of variables a, b, and c are all the same, print the message “All equal” on the screen and find their sum.

Programming Exercise

- Write a C program that reads in three numbers, checks whether they can be the lengths of the three sides of a triangle (the length of each side must be less than the sum of the other two); and finally determines whether the triangle is scalene, isosceles, equilateral, or right-angled.
 - *Isosceles triangle*: only two sides of the triangle are of equal length;
 - *Equilateral triangle*: all three sides are of equal length;
 - *Right-angled triangle*: the square of the longest side is equal to the summation of the squares of the other two sides;
 - *Scalene triangle*: any triangle which do not meet any of the criteria above.



REPETITION STRUCTURE

The Essentials of Repetition

- Loop
 - Group of instructions computer executes repeatedly while some condition remains true
- Counter-controlled repetition
 - Definite repetition: know how many times loop will execute
 - Control variable used to count repetitions
- Sentinel-controlled repetition
 - Indefinite repetition
 - Used when number of repetitions not known
 - Sentinel value indicates "end of data"

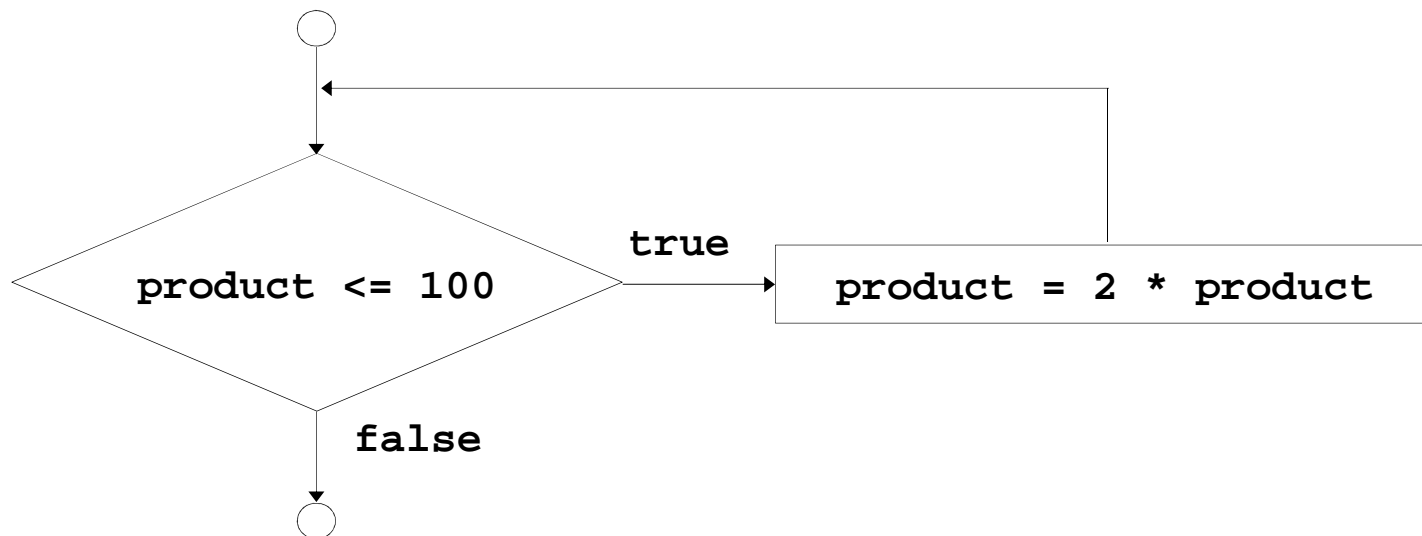
The `while` Repetition Structure

- Repetition structure
 - Programmer specifies an action to be repeated while some condition remains **true**
 - e.g.:
 - While there are more items on my shopping list*
 - Purchase next item and cross it off my list*
 - **while** loop repeated until condition becomes **false**

The `while` Repetition Structure

- Example:

```
int product = 2;  
while ( product <= 100 )  
    product = 2 * product;
```



Example: Counter-Controlled Repetition

- A class of 10 students took a quiz. The grades (integers in the range 0 to 100) for this quiz are available to you. Determine the class average on the quiz
- The algorithm
 - Set total to zero*
 - Set grade counter to one*
 - While grade counter is less than or equal to 10*
 - Input the next grade*
 - Add the grade into the total*
 - Add one to the grade counter*
 - Set the class average to the total divided by ten*
 - Print the class average*

```
/* Class average program with counter-controlled repetition */
#include <stdio.h>

int main()
{
    int counter, grade, total, average;

    /* initialization phase */
    total = 0;
    counter = 1;

    /* processing phase */
    while ( counter <= 10 ) {
        printf( "Enter grade: " );
        scanf( "%d", &grade );
        total = total + grade;
        counter = counter + 1;
    }

    /* termination phase */
    average = total / 10.0;
    printf( "Class average is %d\n", average );

    return 0;    /* indicate program ended successfully */
}
```

```
Enter grade: 98
Enter grade: 76
Enter grade: 71
Enter grade: 87
Enter grade: 83
Enter grade: 90
Enter grade: 57
Enter grade: 79
Enter grade: 82
Enter grade: 94
Class average is 81
```

A Similar Problem

- Problem becomes:

Develop a class-averaging program that will process an arbitrary number of grades each time the program is run.

- Unknown number of students
 - How will the program know to end?
-
- Use sentinel value
 - Also called signal value, dummy value, or flag value
 - Indicates “end of data entry.”
 - Loop ends when user inputs the sentinel value
 - Sentinel value chosen so it cannot be confused with a regular input (such as **-1** in this case)

Formulating Algorithms with Top-Down, Stepwise Refinement

- Top-down, stepwise refinement
 - Begin with a pseudocode representation of the *top*:
Determine the class average for the quiz
 - Divide *top* into smaller tasks and list them in order:
Initialize variables
Input, sum and count the quiz grades
Calculate and print the class average
- Many programs have three phases:
 - Initialization: initializes the program variables
 - Processing: inputs data values and adjusts program variables accordingly
 - Termination: calculates and prints the final results

```

/* Class average program with sentinel-controlled repetition */
#include <stdio.h>
int main()
{
    float average;
    int counter, grade, total;

    /* initialization phase */
    total = 0;
    counter = 0;

    /* processing phase */
    printf( "Enter grade, -1 to end: " );
    scanf( "%d", &grade );
    while ( grade != -1 )
    {
        total = total + grade;
        counter = counter + 1;
        printf( "Enter grade, -1 to end: " );
        scanf( "%d", &grade );
    }
    /* termination phase */
    if( counter != 0 ) {
        average = ( float ) total / counter;
        printf( "Class average is %.2f", average ); }
    else
        printf( "No grades were entered\n" );
    return 0;    /* indicate program ended successfully */
}

```

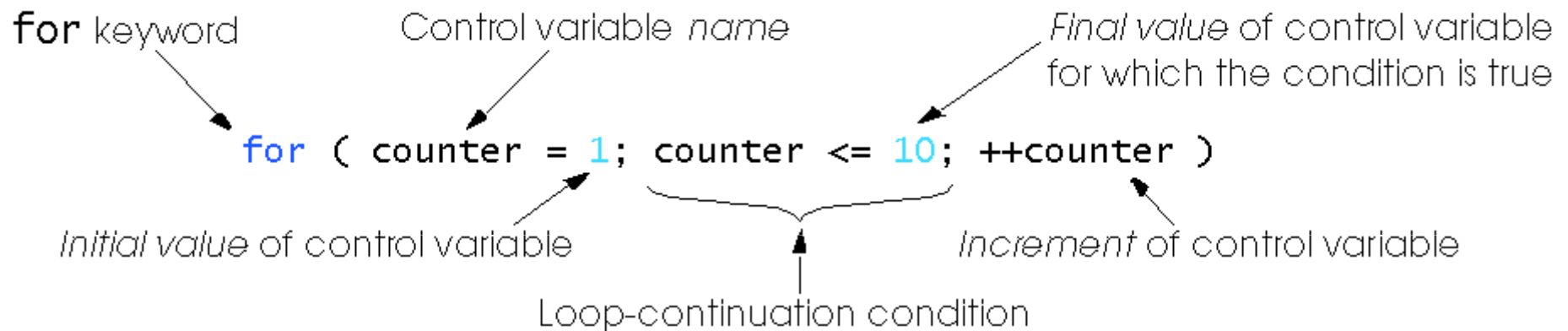
```

Enter grade, -1 to end: 75
Enter grade, -1 to end: 94
Enter grade, -1 to end: 97
Enter grade, -1 to end: 88
Enter grade, -1 to end: 70
Enter grade, -1 to end: 64
Enter grade, -1 to end: 83
Enter grade, -1 to end: 89
Enter grade, -1 to end: -1
Class average is 82.50

```

The `for` Repetition Structure

- Format when using `for` loops



- Example:

```
for(counter = 1; counter <= 10; counter++ )  
    printf( "%d\n", counter );
```

- Prints the integers from one to ten

No
semicolon
(;) after for
statement

The `for` Repetition Structure

- For loops can usually be rewritten as while loops:

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

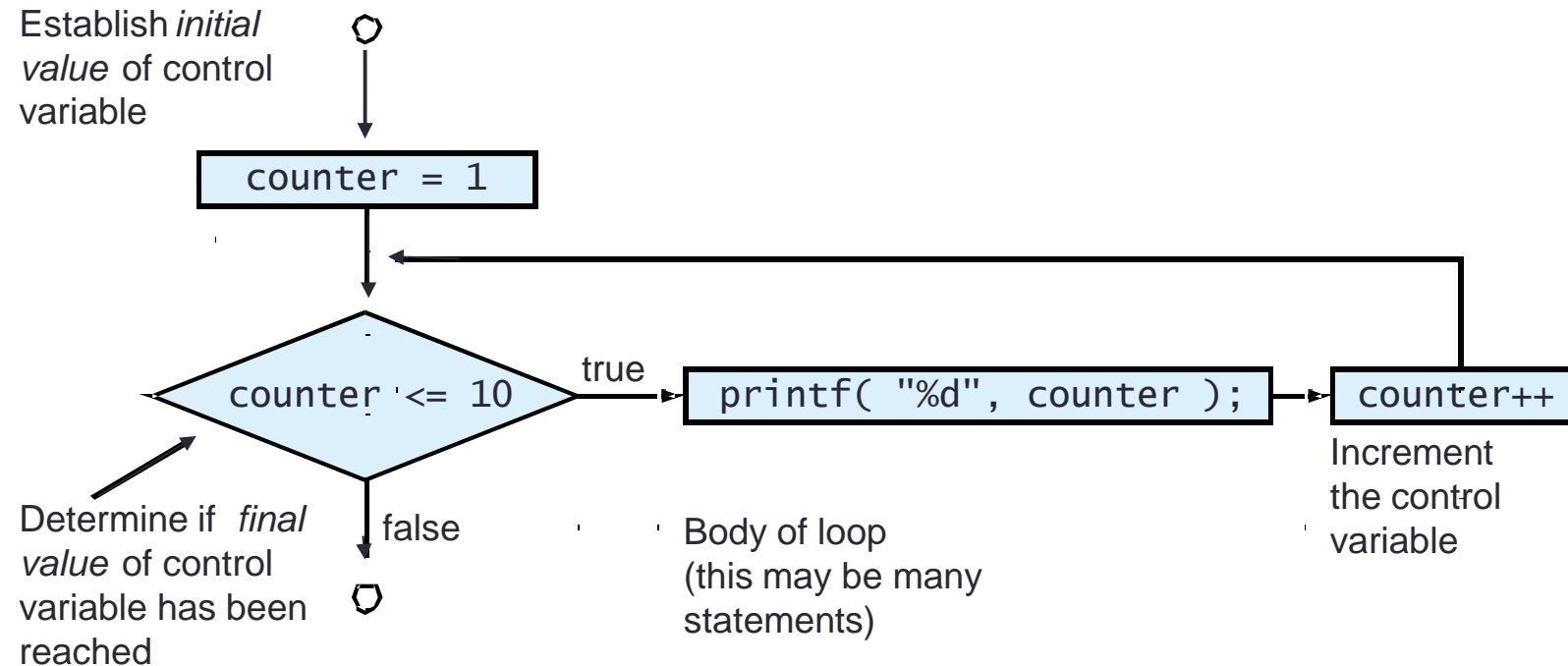
- Initialization and increment

- Can be comma-separated lists

```
for ( i = 0, j = 0; j + i <= 10; j++, i++ )
    printf( "%d\n", j + i );
```

- Initialization, loop-continuation, and increment can contain arithmetic expressions. If *x* equals 2 and *y* equals 10
`for (j = x; j <= 4 * x * y; j += y / x)` equals to
`for (j = 2; j <= 80; j += 5)`

The for Flowchart



Write a program that prints the sum of all numbers from 2 to 100

```
/*Summation with for */
#include <stdio.h>

int main()
{
    int sum = 0, number;
    for ( number = 2; number <= 100; number += 1 )
        sum += number;
    printf( "Sum is %d\n", sum );
    return 0;
}
```

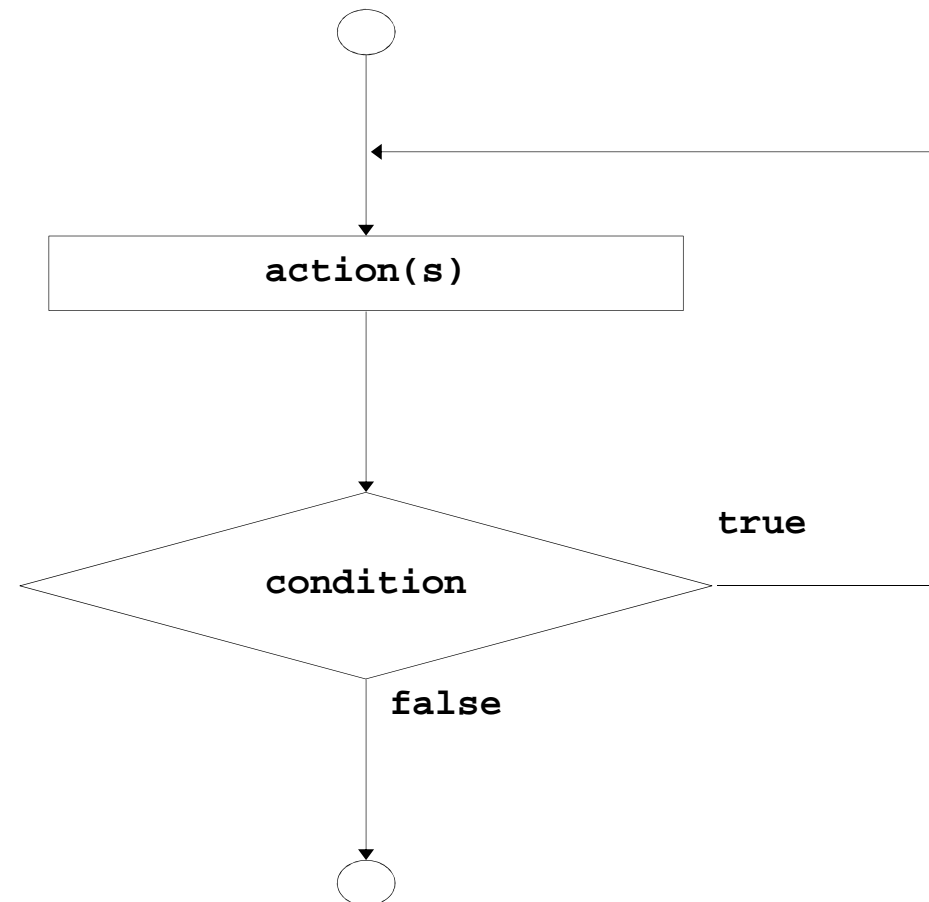
Sum is 2550

The `do/while` Repetition Structure

- The `do/while` repetition structure
 - Similar to the `while` structure
 - Condition for repetition tested after the body of the loop is performed
 - All actions are performed at least once
 - Format:

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

The do/while Repetition Structure



Prints the integers from one to ten

```
/*Using the do/while repetition structure */
```

```
#include <stdio.h>
int main()
{
    int counter = 1;

    do {
        printf( "%d  ", counter );
        counter = counter + 1;
    } while ( counter <= 10 );

    return 0;
}
```

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

Nested control structures

- Problem
 - A college has a list of test results (1 = pass, 2 = fail) for 10 students
 - Write a program that analyzes the results
 - If more than 8 students pass, print "Raise Tuition"
- Notice that
 - The program must process 10 test results
 - Counter-controlled loop will be used
 - Two counters can be used
 - One for number of passes, one for number of fails
 - Each test result is a number—either a 1 or a 2
 - If the number is not a 1, we assume that it is a 2

```
/* Class average program with sentinel-controlled repetition */
#include <stdio.h>
int main()
{
    int passes = 0;
    int failures = 0;
    int student = 1;
    int result;

    while(student <= 10)
    {
        printf( "Enter result: 1(Pass), 2(Fail): " );
        scanf( "%d", &result);

        if(result == 1)
            passes++;
        else
            failures++;

        student = student + 1;
    }

    printf("Passed:%d\n", passes);
    printf("Failed:%d\n", failures );

    if(passes > 8)
        printf("Raise the tuition!");

    return 0;    /* indicate program ended successfully */
}
```

An Example Run

Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1

Passed 9

Failed 1

Raise tuition

Nested Loops

- When a loop body includes another loop construct this is called a *nested loop*.
- In a nested loop structure the inner loop is executed from the beginning every time the body of the outer loop is executed.

```
value = 0;  
for (i=1; i<=10; i=i+1)  
    for (j=1; j<=5; j=j+1)  
        value = value + 1;
```

- How many times the inner loop is executed?

Printing a triangle

- Write a program to draw a triangle like the following:
(input: the number of lines)

*

* *

* * *

* * * *

* * * * *

We can use a nested for-loop:

```
for (i=1; i<=num_lines; ++i)
{
    for (j=1; j<=i; ++j)
        printf("*");
    printf("\n");
}
```

Nesting while and for

```
int main()
{
    int num, count, total = 0;

    printf("Enter a value or a negative number to end: " );
    scanf("%d", &num );

    while( num >= 0 ) {
        for (count = 1; count <= num; count++)
            total = total + count;
        printf("%d %d", num, total);
        printf( "Enter a value or a negative number to end:");
        scanf( "%d", &num );
        total = 0;
    }
    return 0;
}
```

This program reads numbers until the user enters a negative number. For each number read, it prints the number and the summation of all values between 1 and the given number.

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    char grade;
```

```
    int aCount = 0, bCount = 0, cCount = 0,
```

```
        dCount = 0, eCount = 0 ;
```

```
    printf( "Enter the letter grades. Enter X to exit. \n" );
```

```
    while((grade = getchar()) != 'X')
```

```
    {
```

```
        switch ( grade ) {
```

```
            case 'A': ++aCount;
```

```
                    break;
```

```
            case 'B': ++bCount;
```

```
                    break;
```

```
            case 'C': ++cCount;
```

```
                    break;
```

```
            case 'D': ++dCount;
```

```
                    break;
```

```
            case 'F': ++fCount;
```

```
                    break;
```

```
            default:          /* catch all other characters */
```

```
                printf( "Incorrect letter grade entered." );
```

```
                printf( "Enter a new grade.\n" );
```

```
                break;
```

```
        }
```

```
    }
```

```
printf("Total for each letter grade are:\n" );
printf("A: %d\n", aCount );
printf("B: %d\n", bCount );
printf("C: %d\n", cCount );
printf("D: %d\n", dCount );
printf("E: %d\n", eCount );
printf("F: %d\n", gfCount );

return 0;
}
```

Sample Output:

Enter the letter grades. Enter X to exit.

A

B

C

C

A

F

C

E

Incorrect letter grade entered. Enter a new grade.

D

X

Totals for each letter grade are:

A: 2

B: 1

C: 3

D: 1

F: 1

The break and continue Statements

- break
 - Causes immediate exit from a while, for, do...while or switch statement
 - Program execution continues with the first statement after the structure
 - Common uses of the break statement
 - Escape early from a loop
 - Skip the remainder of a switch statement

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int x;
```

```
    for(x = 1; x <= 10 ; x++)
```

```
    {
```

```
        if( x == 5) {
```

```
            break;
```

```
            printf("%d ", x);
```

```
        }
```

```
        printf("\nBroke out of the loop at x=%d ", x);
```

```
        return 0;
```

```
}
```

```
1 2 3 4
```

```
Broke out of loop at x == 5
```

The break and continue Statements

- `continue`

- Skips the remaining statements in the body of a `while`, `for` or `do...while` statement
 - Proceeds with the next iteration of the loop

- `while` and `do...while`

- Loop-continuation test is evaluated immediately after the `continue` statement is executed

- `for`

- Increment expression is executed, then the loop-continuation test is evaluated

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int x;
```

```
    for(x = 1; x <= 10 ; x++)
```

```
    {
```

```
        if( x == 5) {
```

```
            continue;
```

```
            printf("%d ", x);
```

```
        }
```

```
    printf("\nUsed continue to skip printing the value 5");
```

```
    return 0;
```

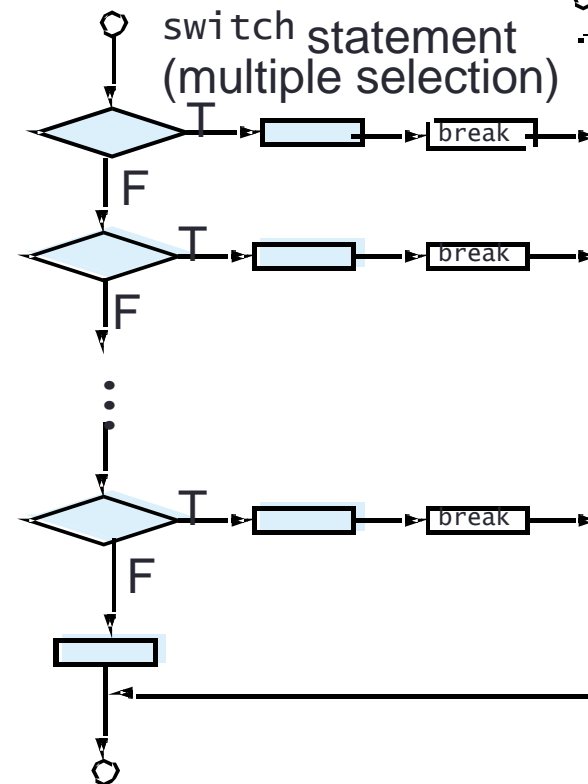
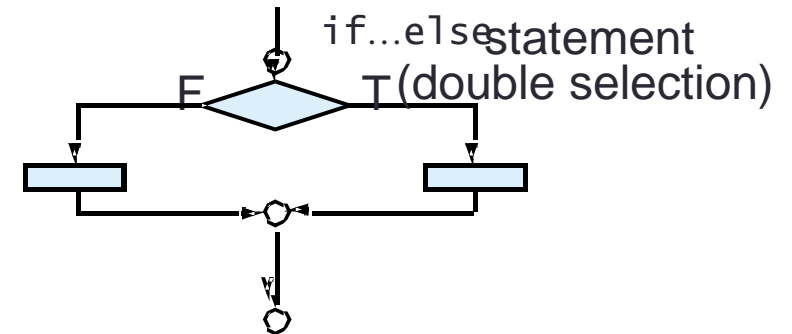
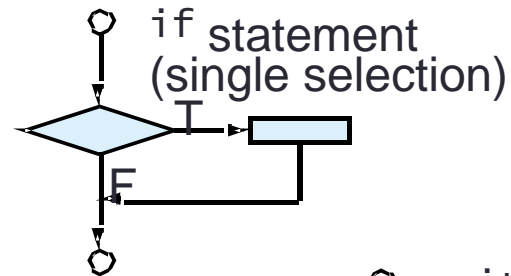
```
}
```

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

```
Used continue to skip printing the value 5
```

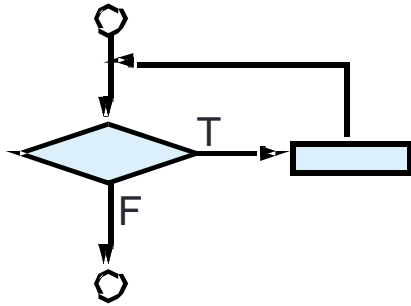

Structured-Programming Summary

Sequence

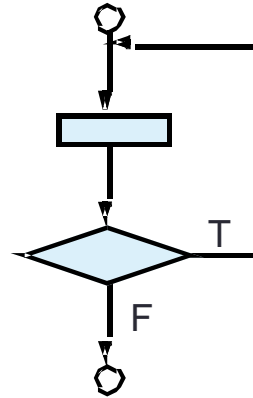


Structured-Programming Summary

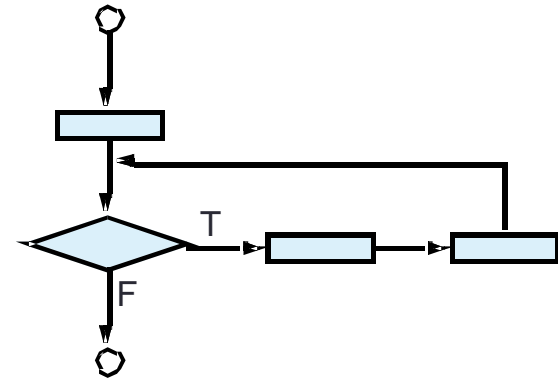
while statement



do..while statement



for statement



Programming Exercises

1. Write a program which calculates the factorial of a number entered by the user
2. Write a program which prints the fibonacci numbers until the n^{th} number in the sequence (n is entered by the user)
3. Write a program which ask the user to enter two numbers and one operator (*, /, +, -, %), and prints the result of the selected operation on these numbers.