CMPE 252
C PROGRAMMING

SPRING 2021

WEEK 2-3

SELECTION STRUCTURES: IF AND SWITCH STATEMENTS

CHAPTER 4

Problem Solving & Program Design in C

Eighth Edition

Global Edition

Jeri R. Hanly & Elliot B. Koffman

Control Structures

- control structure
  - a combination of individual instructions into a single logical unit with one entry point and one exit point
- compound statement
  - a group of statements bracketed by { and } that are executed sequentially

Instructions are organized into three kinds of control structures to control the execution flow :
- sequence (compound statements are used to specify sequential flow)
- selection structures (e.g. if then else)
- repetition structures (loops)

Compound Statement

```
{
        statement;
        statement;
                .
                .
                .
        statement;
}
```

Used to specify sequential flow

Selection Structures

What if we want / need to add decision in our program ?

-> We use : if / else statements

```
if   (x != 0)
   product = product / x;
```

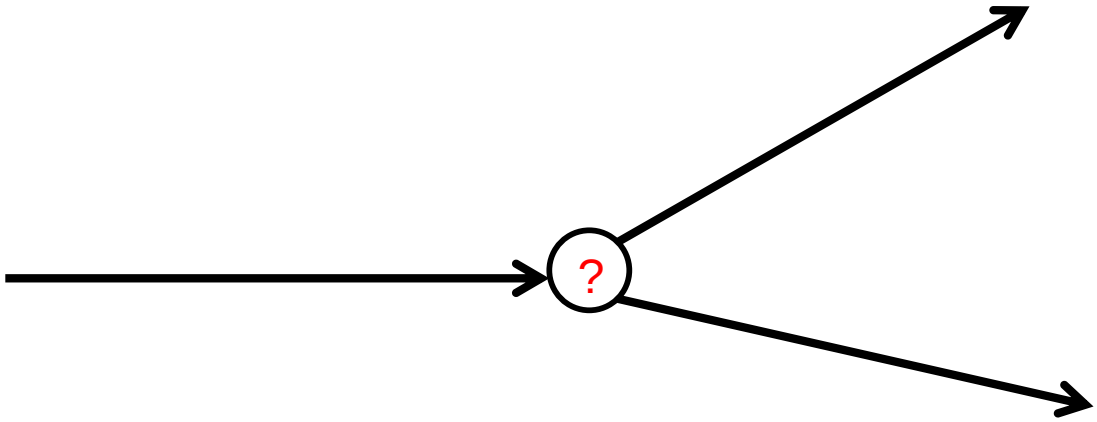
İf statement with one alternative

```
if   (x != 0)
   product = product / x;
else
   printf("I can not
divide, since divisor is =
0 \n");
```

İf statement with two alternatives

Control Structures

- selection control structure
  - a control structure that chooses among alternative program statements



Conditions

- an expression that is either FALSE
  - represented by 0
- or TRUE
  - usually represented by 1

my_age > 40

In ANSI C , there is NO «bool» type,

anything that is **NOT«0»** is **TRUE**

Relational and Equality Operators

Operator	Meaning	Type
<	less than	relational
>	greater than	relational
<=	less than or equal to	relational
>=	greater than or equal to	relational
==	equal to	equality
!=	not equal to	equality

variable relational-operator variable
variable relational-operator constant
variable equality-operator variable
variable equality-operator constant

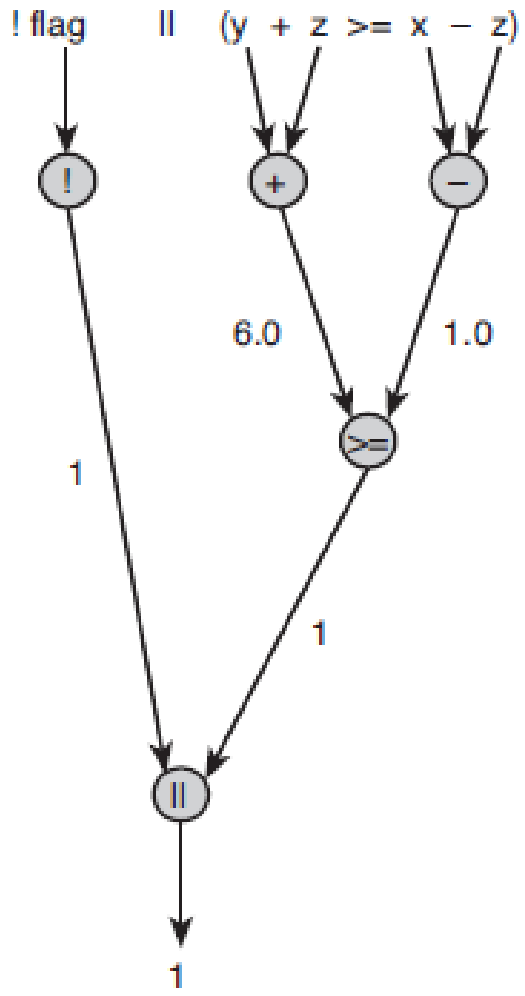
Logical Operators

- logical expressions
  - an expression that uses one or more of the logical operators
    - && (AND)
    - || (OR)
    - ! (NOT)

Operator Precedence

Operator	Precedence
function calls	highest (evaluated first)
! + - & (unary operator)	
* / %	
+ -	
< <= >= >	
== !=	
&&	
||	
=	lowest (evaluated last)

Figure 4.1
Evaluation Tree and Step-by-Step Evaluation for !flag || (y + z >= x - z)



flag	y	z	x
0	4.0	2.0	3.0

```
! flag   ||   (y   +   z      >=      x   -   z)

  0          4.0   2.0              3.0   2.0

  1          6.0                    1.0

                          1

  1
```


Short-Circuit Evaluation

- stopping evaluation of a logical expression as soon as its value can be determined

```
!flag || (y + z >= x - z)
```

An expression of the form *a* || *b* must be true if *a* is true. Consequently, C stops evaluating the expression when it determines that the value of `!flag` is `1` (true).

Similarly, an expression of the form *a* && *b* must be false if *a* is false, so C would stop evaluating such an expression if its first operand evaluates to 0.

Short-Circuit Evaluation

- We can use short-circuit evaluation to prevent potential run-time errors.
  - e.g. (num % div == 0) – what if div ==0 ?
- In this case, the remainder calculation would cause a *division by zero* run-time error.
- However, we can prevent this error by using the revised condition
- (div != 0 && (num % div == 0))

Writing English Conditions in C

EXAMPLE 4.3 Table 4.7 shows some English conditions and the corresponding C expressions. Each expression is evaluated assuming `x` is `3.0`, `y` is `4.0`, and `z` is `2.0`.

TABLE 4.7 English Conditions as C Expressions

English Condition	Logical Expression	Evaluation
`x` and `y` are greater than `z`	`x > z && y > z`	`1 && 1` is 1 (true)
`x` is equal to `1.0` or `3.0`	`x == 1.0 || x == 3.0`	`0 || 1` is 1 (true)
`x` is in the range `z` to `y`, inclusive	`z <= x && x <= y`	`1 && 1` is 1 (true)
`x` is outside the range `z` to `y`	`!(z <= x && x <= y)` `z > x || x > y`	`!(1 && 1)` is 0 (false) `0 || 0` is 0 (false)

Comparing Characters

Expression	Value
‘9’ >= ‘0’	1 (true)
‘a’ < ‘e’	1 (true)
‘B’ <= ‘A’	0 (false)
‘Z’ == ‘z’	0 (false)
‘a’ <= ‘A’	System dependent
‘a’ <= ch && ch <= ‘z’	1 (true) if ch is a lowercase letter

THE IF-STATEMENT

making decisions

if-statement with one alternative

```
if  (x != 0)
        product = product * x;
```

if-statement with two alternatives

```
if  (rest_heart_rate > 75)
        printf(“Keep up your exercise program!\n”);
else
        printf(“Your hear is doing well!\n”);
```

```c
#include <stdio.h>

int main(void)
{
        int pulse;             /* resting pulse rate for 10 secs */
        int rest_heart_rate;   /* resting heart rate for 1 minute */

        /* Enter your resting pulse rate */
        printf("Take your resting pulse for 10 seconds.\n");
        printf("Enter your pulse rate and press return> ");
        scanf("%d", &pulse);

        /* Calculate resting heart rate for minute */
        rest_heart_rate = pulse * 6;
        printf("Your resting heart rate is %d.\n", rest_heart_rate);

        /* Display message based on resting heart rate */
        if (rest_heart_rate > 75)
            printf("Keep up your exercise program!\n");
        else
            printf("Your heart is in doing well!\n");

        return (0);
}
```

Nested if-statements with more than one variable

```
if (road_status == ‘S’)
        if (temp > 0) {
                printf(“Wet roads ahead\n”);
                printf(“Stopping time doubled\n”);
        }  else  {
                printf(“Icy roads ahead\n”);
                printf(“Stopping time quadrupled\n”);
        }
else
        printf(“Drive carefully!\n”)
```

The switch statement

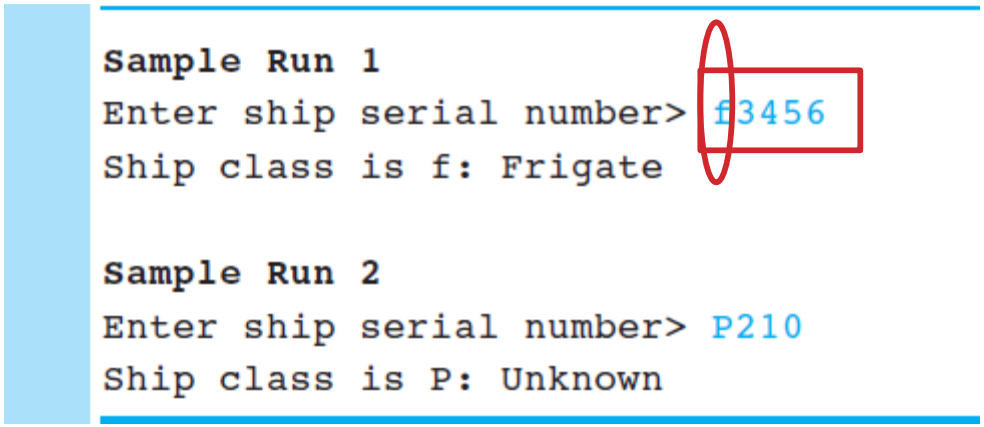
- also used to select one of several alternatives
- useful when the selection is based on the value of
  - a single variable
  - or a simple expression
- values may of type int or char
  - not double

Syntax

```
switch (controlling expression) {
        label set1
                statements1
                break;
        label set2
                statements2
                break;
                .
                .
                .
        label setn
                statementsn
                break;
```

```c
/* FIGURE 4.13  Program Using a switch Statement for Selection */
/*
 * Reads serial number and displays class of ship
 */
#include <stdio.h>
int main(void)
{
    char class;     /* input - character indicating class of ship */

    /* Read first character of serial number */
    printf("Enter ship serial number> ");
    scanf("%c", &class);         /* scan first letter */

    /* Display first character followed by ship class */
    printf("Ship class is %c: ", class);
    switch (class) {
    case 'B':
    case 'b':
            printf("Battleship\n");
            break;
    case 'C':
    case 'c':
            printf("Cruiser\n");
            break;
    case 'D':
    case 'd':
            printf("Destroyer\n");
            break;
    case 'F':
    case 'f':
            printf("Frigate\n");
            break;
    default:
            printf("Unknown\n");
    }

    return (0);
}
```

Figure 4.13
Program Using a *switch* Statement for Selection (cont.)



```
Sample Run 1
Enter ship serial number> f3456
Ship class is f: Frigate

Sample Run 2
Enter ship serial number> P210
Ship class is P: Unknown
```

REPETITION AND LOOP STATEMENTS
CHAPTER 5

Problem Solving & Program Design in C

Eighth Edition

Global Edition

Jeri R. Hanly & Elliot B. Koffman

Repetition in Programs

- loop
  - a control structure that repeats a group of steps in a program
- loop body
  - the statements that are repeated in the loop

while Statement Syntax

```
while (loop repetition condition)
        statement;
```

```
/* display N asterisks. */
count_star = 0
while (count_star  <  N) {
        printf("*");
        count_star = count_star + 1;
}
```

```c
/* FIGURE 5.4  Program to Compute Company Payroll */
/* Compute the payroll for a company */

#include <stdio.h>

int main(void)
{
     double total_pay;      /* company payroll         */
     int    count_emp;      /* current employee        */
     int    number_emp;     /* number of employees     */
     double hours;          /* hours worked            */
     double rate;           /* hourly rate             */
     double pay;            /* pay for this period     */

    /* Get number of employees. */
    printf("Enter number of employees> ");
    scanf("%d", &number_emp);

    /* Compute each employee's pay and add it to the payroll. */
    total_pay = 0.0;
    count_emp = 0;
    while (count_emp < number_emp) {
        printf("Hours> ");
        scanf("%lf", &hours);
        printf("Rate > $");
        scanf("%lf", &rate);
        pay = hours * rate;
        printf("Pay is $%6.2f\n\n", pay);
        total_pay = total_pay + pay;              /* Add next pay. */
        count_emp = count_emp + 1;
    }
    printf("All employees processed\n");
    printf("Total payroll is $%8.2f\n", total_pay);

    return (0);
}
```

```
Enter number of employees> 3
Hours> 50
Rate > $5.25
Pay is $262.50

Hours> 6
Rate > $5.00
Pay is $ 30.00

Hours> 15
Rate > $7.00
Pay is $105.00

All employees processed
Total payroll is $  397.50
```

The for Statement Syntax

for (*initialization expression*;
loop repetition condition;
update expression)
statement;

```
/* Display N asterisks. */
for  (count_star = 0; count_star < N; count_star += 1)
     printf("*");
```

```
3   total_pay = 0.0;
4   for (count_emp = 0;                  /* initialization                */
5        count_emp < number_emp;         /* loop repetition condition     */
6        count_emp += 1) {               /* update                        */
7       printf("Hours> ");
8       scanf("%lf", &hours);
9       printf("Rate > $");
10      scanf("%lf", &rate);
11      pay = hours * rate;
12      printf("Pay is $%6.2f\n\n", pay);
13      total_pay = total_pay + pay;
14  }
```


Increment and Decrement Operators

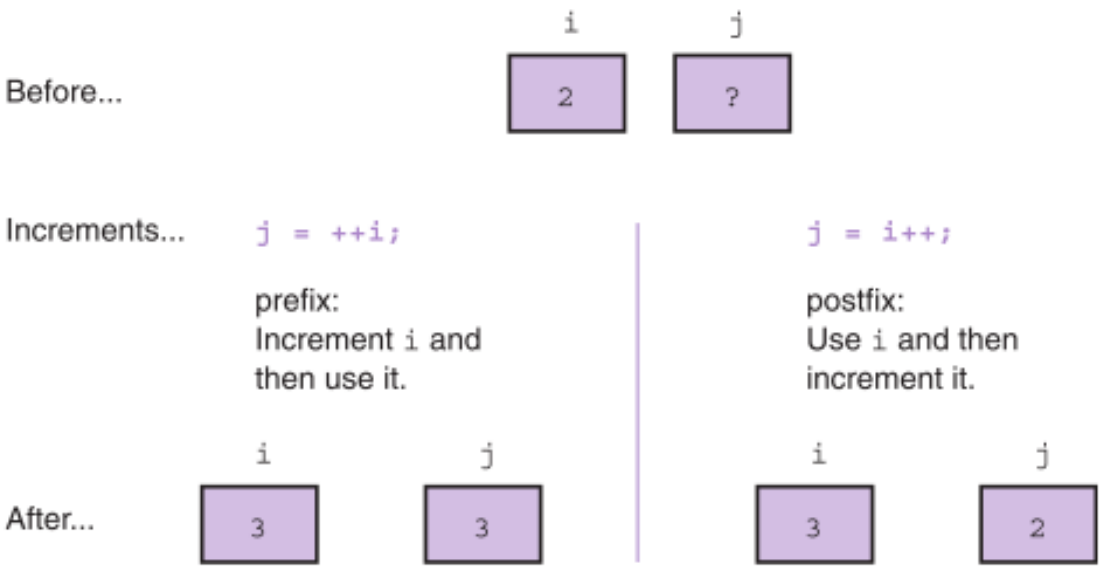
- counter = counter + 1
  count += 1
  counter++ → Postfix increment
  ++counter → Prefix increment
- counter = counter - 1
  count -= 1
  counter-- → Postfix decrement
  --counter → Prefix decrement

Increment and Decrement Operators

- side effect
  - a change in the value of a variable as a result of carrying out an operation

FIGURE 5.6

Comparison of Prefix and Postfix Increments



Conditional Loops

- used when there are programming conditions when you will not be able to determine the exact number of loop repetitions before loop execution begins
- Quick check with for loop:

```
Number of barrels currently in tank> 8500.5
8500.50 barrels are available.

Enter number of gallons removed> 5859.0
After removal of 5859.00 gallons (139.50 barrels),
8361.00 barrels are available.

Enter number of gallons removed> 7568.4
After removal of 7568.40 gallons (180.20 barrels),
8180.80 barrels are available.

Enter number of gallons removed> 8400.0
After removal of 8400.00 gallons (200.00 barrels),
only 7980.80 barrels are left.

*** WARNING ***
Available supply is less than 10 percent of tank's
80000.00-barrel capacity.
```

```c
#include <stdio.h>
#define CAPACITY 80000.0 /* number of barrels tank can hold      */
#define MIN_PCT  10      /* warn when supply falls below this percent of capacity */
#define GALS_PER_BRL 42.0  /* number of U.S. gallons in one barrel */

/* Function prototype */
double monitor_gas(double min_supply, double start_supply);

int main(void)
{
    double start_supply, /* input - initial supply in barrels*/
        min_supply,   /* minimum number of barrels left without warning*/
        current;      /* output - current supply in barrels*/

        /* Compute minimum supply without warning*/
        min_supply = MIN_PCT / 100.0 * CAPACITY;

        /* Get initial supply */
        printf("Number of barrels currently in tank> ");
        scanf("%lf", &start_supply);

        /* Subtract amounts removed and display amount remaining
           as long as minimum supply remains.*/
       current = monitor_gas(min_supply, start_supply);
```

```
26          /* Issue warning*/
27          printf("only %.2f barrels are left.\n\n", current);
28          printf("*** WARNING ***\n");
29
30          printf("Available supply is less than %d percent of tank's\n",
31                  MIN_PCT);
32          printf("%.2f-barrel capacity.\n", CAPACITY);
33
34          return (0);
35  }
```

```
37  /*
38   * Computes and displays amount of gas remaining after each delivery
39   * Pre : min_supply and start_supply are defined.
40   * Post: Returns the supply available (in barrels) after all permitted
41   *       removals. The value returned is the first supply amount that is
42   *       less than min_supply.
43   */
44  double monitor_gas(double min_supply, double start_supply)
45  {
46          double remov_gals, /* input - amount of current delivery         */
47                 remov_brls, /*         in barrels and gallons             */
48                 current;    /* output - current supply in barrels         */
49
50          for (current = start_supply;current >= min_supply;current -= remov_brls)
51          {
52             printf("%.2f barrels are available.\n\n", current);
53             printf("Enter number of gallons removed> ");
54             scanf("%lf", &remov_gals);
55             remov_brls = remov_gals / GALS_PER_BRL;
56
57             printf("After removal of %.2f gallons (%.2f barrels),\n",remov_gals, remov_brls);
58          }
59
60          return (current);
61  }
```

Loop Design

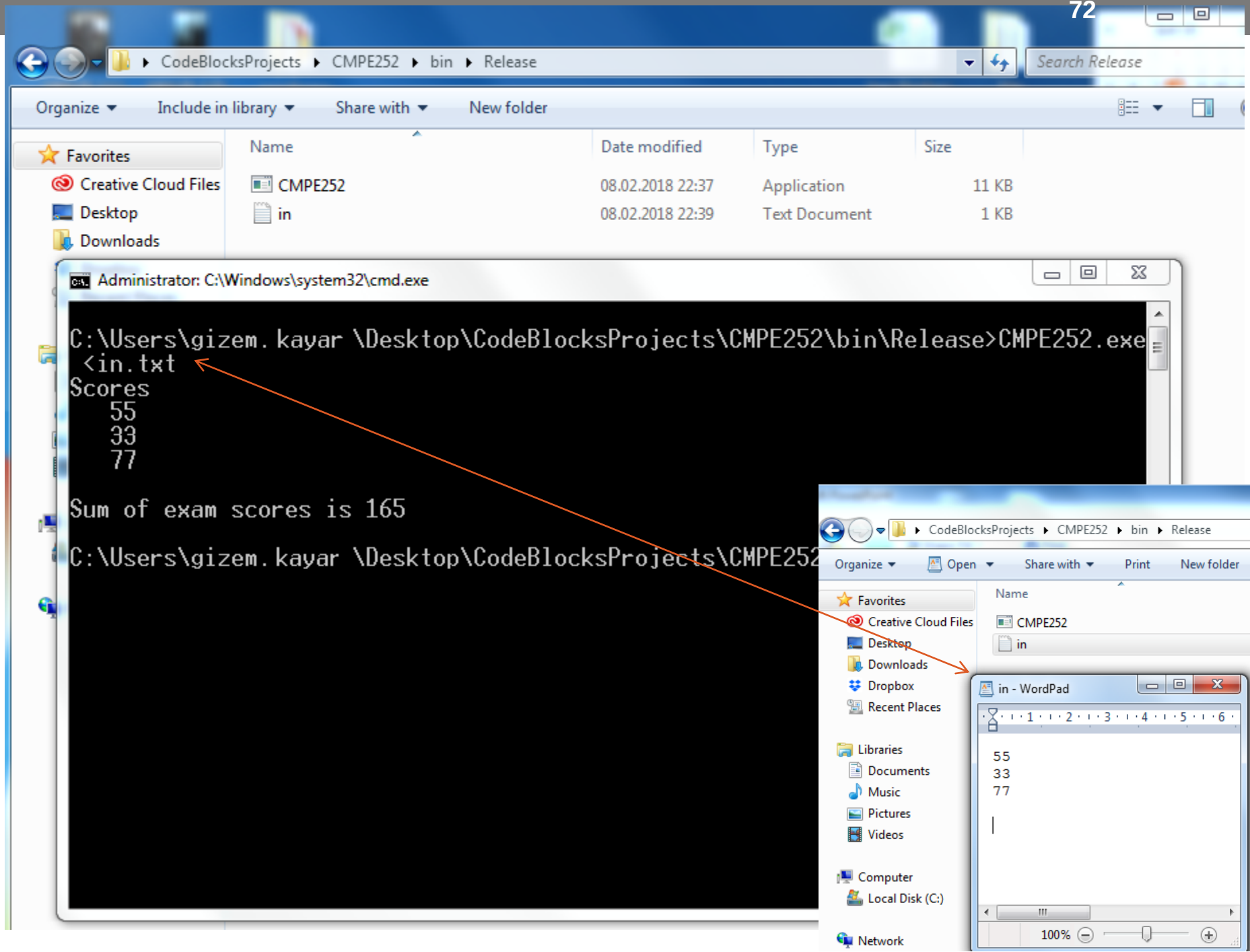
- Sentinel-Controlled Loops
  - sentinel value: an end marker that follows the last item in a list of data
- Endfile-Controlled Loops
- Infinite Loops on Faulty Data

```c
#include <stdio.h>
#define SENTINEL -99

int main(void)
{
        int sum = 0, /* output - sum of scores input so far        */
            score;   /* input - current score                      */

        /* Accumulate sum of all scores.                           */
        printf("Enter first score (or %d to quit)> ", SENTINEL);
        scanf("%d", &score);      /* Get first score.              */
        while (score != SENTINEL)
        {
            sum += score;
            printf("Enter next score (%d to quit)> ", SENTINEL);
            scanf("%d", &score);   /* Get next score.            */
        }
        printf("\nSum of exam scores is %d\n", sum);

        return (0);
}
```

Endfile-Controlled Loop Design

1. Get the first *data value* and save *input status*
2. while *input status* does not indicate that end of file has been reached
    3. Process *data value*
    4. Get next *data value* and save *input status*

```c
#include <stdio.h>

int main(void)
{
	int sum = 0,	/* sum of scores input so far */
	    score,	/* current score */
	    input_status; /* status value returned by scanf */

	printf("Scores\n");

	input_status = scanf("%d", &score);
	while (input_status != EOF)
	{
		printf("%5d\n", score);
		sum += score;
		input_status = scanf("%d", &score);
	}

	printf("\nSum of exam scores is %d\n", sum);

	return (0);
}
```


CodeBlocksProjects ▸ CMPE252 ▸ bin ▸ Release
Search Release
Organize
Include in library
Share with
New folder
Favorites
Creative Cloud Files
Desktop
Downloads
Name
Date modified
Type
Size
CMPE252
08.02.2018 22:37
Application
11 KB
in
08.02.2018 22:39
Text Document
1 KB
Administrator: C:\Windows\system32\cmd.exe
C:\Users\gizem.kayar \Desktop\CodeBlocksProjects\CMPE252\bin\Release>CMPE252.exe
<in.txt
Scores
55
33
77
Sum of exam scores is 165
C:\Users\gizem.kayar \Desktop\CodeBlocksProjects\CMPE252
CodeBlocksProjects ▸ CMPE252 ▸ bin ▸ Release
Organize
Open
Share with
Print
New folder
Favorites
Creative Cloud Files
Desktop
Downloads
Dropbox
Recent Places
Libraries
Documents
Music
Pictures
Videos
Computer
Local Disk (C:)
Network
Name
CMPE252
in
in - WordPad
55
33
77
100%

Nested Loops

- Loops may be nested just like other control structures
- Nested loops consist of an outer loop with one or more inner loops
- Each time the outer loop is repeated, the inner loops are reentered, their loop control expressions are reevaluated, and all required iterations are performed

```c
#include <stdio.h>

#define SENTINEL 0
#define NUM_MONTHS 12

int main(void)
{
    int month,      /* number of month being processed          */
        mem_sight, /* one member's sightings for this month    */
        sightings; /* total sightings so far for this month    */

    printf("BALD EAGLE SIGHTINGS\n");
    for (month = 1; month <= NUM_MONTHS;++month)
    {
            sightings = 0;
            scanf("%d", &mem_sight);
            while (mem_sight != SENTINEL)
            {
                if (mem_sight >= 0)
                    sightings += mem_sight;
                else
                    printf("Warning, negative count %d ignored\n",
                            mem_sight);
                scanf("%d", &mem_sight);
            }   /* inner while */

            printf(" month %2d: %2d\n", month, sightings);
        }   /* outer for */

        return (0);
}
```

Quick Check

- Write a program that gives the following output (multiplication table from 0 to 9):

```
*  0  1  2  3  4  5  6  7  8  9
0  0  0  0  0  0  0  0  0  0  0
1  0  1  2  3  4  5  6  7  8  9
2  0  2  4  6  8 10 12 14 16 18
3  0  3  6  9 12 15 18 21 24 27
4  0  4  8 12 16 20 24 28 32 36
5  0  5 10 15 20 25 30 35 40 45
6  0  6 12 18 24 30 36 42 48 54
7  0  7 14 21 28 35 42 49 56 63
8  0  8 16 24 32 40 48 56 64 72
9  0  9 18 27 36 45 54 63 72 81
```

```c
#include <stdio.h>
#define NUM_DIGITS  10

int main(void)
{
    int  factor_1, factor_2,product;

       /* Display the table heading.                */
       printf("\n*");
       for (factor_2 = 0; factor_2 < NUM_DIGITS; ++factor_2)
          printf("%3d", factor_2);

       /* Display the table body.*/
       for (factor_1 = 0; factor_1 < NUM_DIGITS; ++factor_1)
       {
          printf("\n%d", factor_1); /* Start a row with first factor. */
          for (factor_2 = 0; factor_2 < NUM_DIGITS; ++factor_2)
          {
             product = factor_1 * factor_2;
             printf("%3d", product);
          }
       }
       printf("\n");

       return (0);
    }

```

do-while Statement

- For conditions where we know that a loop must execute at least one time

1. Get a *data value*
2. If *data value* isn't in the acceptable range, go back to step 1.

do-while Syntax

```
do
        statement;
while (loop repetition condition);
```

```
/* Find first even number input */
do
        status = scanf("%d", &num);
while (status > 0  &&  (num % 2)  !=  0);
```

References

1. Problem Solving & Program Design in C, Jeri R. Hanly & Elliot B. Koffman, Pearson 8. Edition, Global Edition