BLG 102E
Introduction to Scientific Computing and Engineering

SPRING 2025

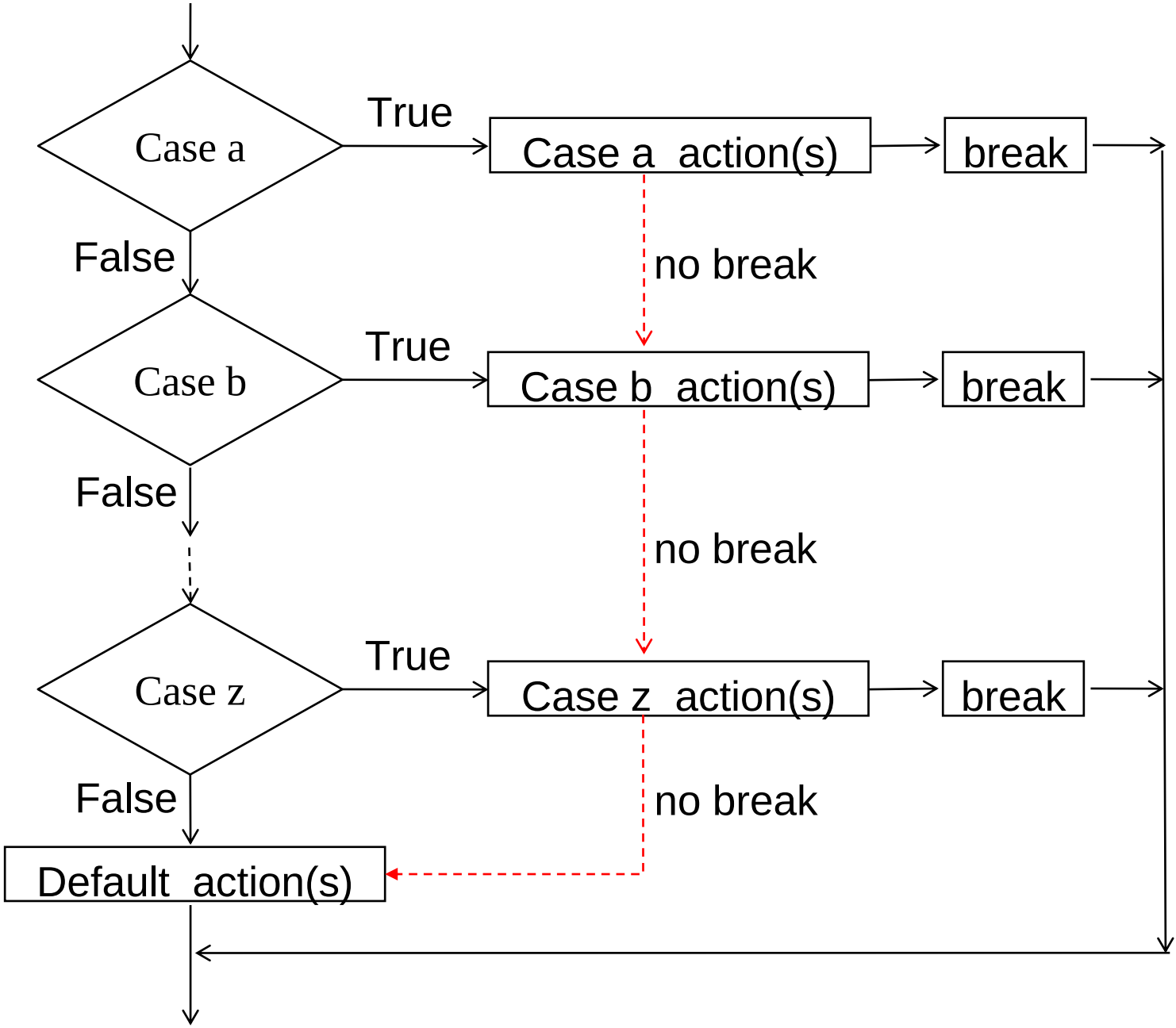
WEEK 4

ISTANBUL TECHNICAL UNIVERSITY

Switch Statement

- Flowchart of the `switch` statement

```
switch (value) {
  case 1:
    actions
  case 2:
    actions

  ...

  default:
    actions
}
```



Days Switch

```
switch (month) {
  case 1:
    days = 31;
    break;
  case 2:
    days = leap ? 29 : 28;
    break;
  case 3:
    days = 31;
    break;

  ...
  ...

  case 11:
    days = 30;
    break;
  case 12:
    days = 31;
    break;
}
```

```
switch (month) {
  case 2:
    days = leap ? 29 : 28;
    break;
  case 4:
  case 6:
  case 9:
  case 11:
    days = 30;
    break;
  default:
    days = 31;
    break;
}
```

Enumerations

- define multiple named constants in sequence
- as a type

```
enum ENUM_NAME { NAME1 = VAL1, NAME2 = VAL2, ...};
```

- variable definition:

```
enum ENUM_NAME VARIABLE_NAME= INITIAL_VALUE;
```

Enumeration Values

- if no value given, previous + 1

```
enum month_e { JAN  = 1, FEB, MAR, . . . , DEC };
```

- if no value given for first item, start from 0

```
enum month_e { JAN,  FEB, MAR, . . . , DEC };
```

Enumeration Example

```
enum month_e {JAN = 1,  FEB = 2, ... , DEC = 12};


enum month_e month = FEB;
```

Days Switch

```
switch (month) {
    case FEB:
        days = leap ? 29 : 28;
        break;
    case APR:
    case JUN:
    case SEP:
    case NOV:
        days = 30;
        break;
    default:
        days = 31;
        break;
}
```

Type Aliases

- give an existing type a new name

```
typedef OLD_NAME NEW_NAME;
```

- both names can be used interchangibly

Type Alias Example

```
enum month_e { JAN = 1, FEB, MAR, APR, ... , DEC };

typedef enum month_e month_t;

month_t month = FEB;
```

Characters

- data type for characters: `char`
- literals written within single quotes

```
char gender = 'F';
```

Character Values

- one-byte numeric value: ASCII number

```
char gender = 'F';

// same as:
char gender = 70;
```

- only reliable with English letters

Character Expressions

- numeric operations allowed

```
letter - 'A'   // order in alphabet

digit - '0'    // convert character digit to number
num + '0'      // convert numeric digit to char
```

I/O

- format specifier: %c

New York Taxi Fare with Distance Unit

- switch-on price:

  $2.50

- per unit distance price:

  $0.50 per 0.2 miles

- *d*: travel distance (in miles)

$$2.50 + 0.50 \cdot \left\lfloor \frac{d}{0.2} \right\rfloor$$

Generalize so that it will work with both km and mile as distance unit!

HOW ???

New York Taxi Fare I/O

- enter distance in miles or km

```
double distance_inp = 0.0;   // travel distance input
char unit;                   // distance unit (k or m)

printf("Enter distance: ");
scanf("%lf %c", &distance_inp, &unit);
```

New York Taxi Fare Program

```
switch (unit) {
    case 'm':
    case 'M':
        distance = distance_inp;
        break;
    case 'k':
    case 'K':
        distance = distance_inp / KM_PER_MILE;
        break;
    default:
        printf("unknown unit");
        return 1;
}
```

Loops
(Repetitions)

Factorial: Algorithm

Write a program to calculate the factorial of a number.

- **INPUT:** An integer number n (where n >= 0).
- **OUTPUT:** Factorial of the n.
- **PROCESSING:** Factorial is computed as the following.

  N ! = 1 * 2 * 3 * 4 * ….. * N
  0! = 1

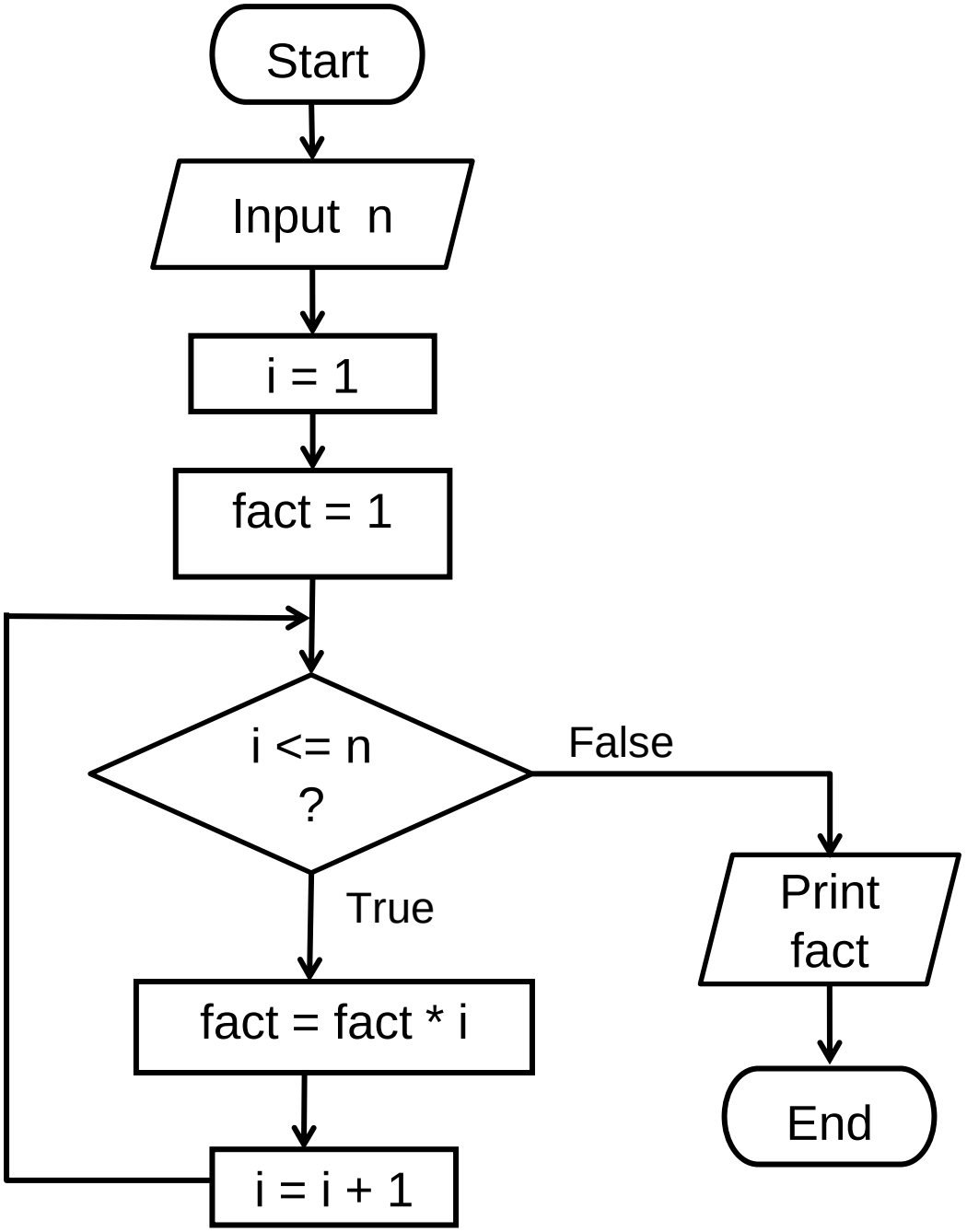
Factorial: Algorithm

Variables:

n : Number (loop limit)

i : Loop counter

fact : Factorial result



Iterative Statement

- execute a block repeatedly based on a condition: loop

```
while (CONDITION) {
    BLOCK
}
```

- stop when the condition becomes false

Factorial: Variables

```
int n = 0;
int i = 0;
int fact = 0;

printf("Enter a number: ");
scanf("%d", &n);
```

Factorial: Iteration

- what should we do at every iteration?
- extend the multiplication and generate the next multiplier

```
fact = fact * i;
i = i + 1;
```

Factorial: Stopping

- How long should we repeat?
- As long as i is less than or equal to n

```
i <= n
```

Factorial: Loop

- iteration:

```
while (i <= n) {
    fact = fact * i;
    i = i + 1;
}
```

- we have to set fact and i before the loop

Factorial: Initial Values

```
i = 1;     // Initialize loop counter
fact = 1; // Initialize factorial

while (i <= n) {
    fact = fact * i;
    i = i + 1;
}
```

Factorial Code

```c
/* A program to calculate factorial. */
#include <stdio.h>

int main()
{
  int n;
  int i;
  int fact;

  printf("Enter a number : ");
  scanf("%d", &n);

  i = 1;    // initialize loop counter
  fact = 1; // initialize factorial

  while (i <= n) {
      fact = fact * i;
      i = i + 1;
  }

  printf("Factorial : %d \n", fact);
  return 0;
}
```

Skipping Loop

- if the condition is false to begin with,

  the block will not be executed at all

- example: incorrect condition

```
while (i > n) {
    fact = fact * i;
    i = i + 1;
}
```

Infinite Loop

- we have to make sure that the condition will eventually be false
- otherwise: infinite loop

- example: decrement i instead of incrementing

```
i = 1;     // Initialize loop counter
fact = 1; // Initialize factorial

while (i <= n) {
     fact = fact * i;
     i = i - 1;
}
```

Flag Variables

- flag: boolean variable to indicate the status of a condition

```
i = 1;     // Initialize loop counter
fact = 1; // Initialize factorial
bool keep_on = true;

while (keep_on) {
    fact = fact * i;
    i = i + 1;
    keep_on = i <= n ? true : false;
}
```

Average Value

- get age values from the user
- report the average of the age values

$$\overline{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

- we don't know how many values (n) the user will enter
- the user will enter 0 for age when finished

  0: sentinel value

Average Value: Variables

```
int age = -1;          // user-entered age value
int age_total = 0;     // total of age values so far
int n = 0;             // number of age values so far
```

Average Value: Iteration

- what do we do at every iteration?

- get the next age value
- add the age value to the age total
- increase the count

```
printf("Age (0 to end): ");
scanf("%d", &age);

age_total += age;
n++;
```

Average Value: Stopping

- when should we stop?
- when age is 0

```
while (age != 0)  {
    printf("Age (0 to end): ");
    scanf("%d", &age);
    age_total  += age;
    n++;
}
```

Average Value: Initial Values

- what should be the initial values for age_total and n?
- both should be zero

Average Value: Result

- average: divide total by count

```
double age_avg = total / n;
```

- integer division

Average Value Code

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

int main() {
    int age = 0;            // user-entered age value
    int age_total = 0;      // total of age values so far
    int n = 0;              // number of age values so far
    double age_avg = 0.0;   // average age

    age_total = 0;
    n = 0;
    age = 1;

    while (age != 0) {
       printf("Age (0 to end): ");
       scanf("%d", &age);
       age_total += age;
       n++;
    }

    age_avg = (double)age_total / n;
    printf("Average age: %.1lf\n", age_avg);
    return 0;
}
```

Average Value Problem

- what if the first value the user types is 0?
- division zero by zero error (runtime error)

```
if (n > 0) {
    age_avg = (double) total / n;
}
```

Average Value: Alternative

- ask the user how many values they will input

```
int n = 0;

printf("Enter number of age values: ");
scanf("%d", &n);
```

Counter Controlled Loops

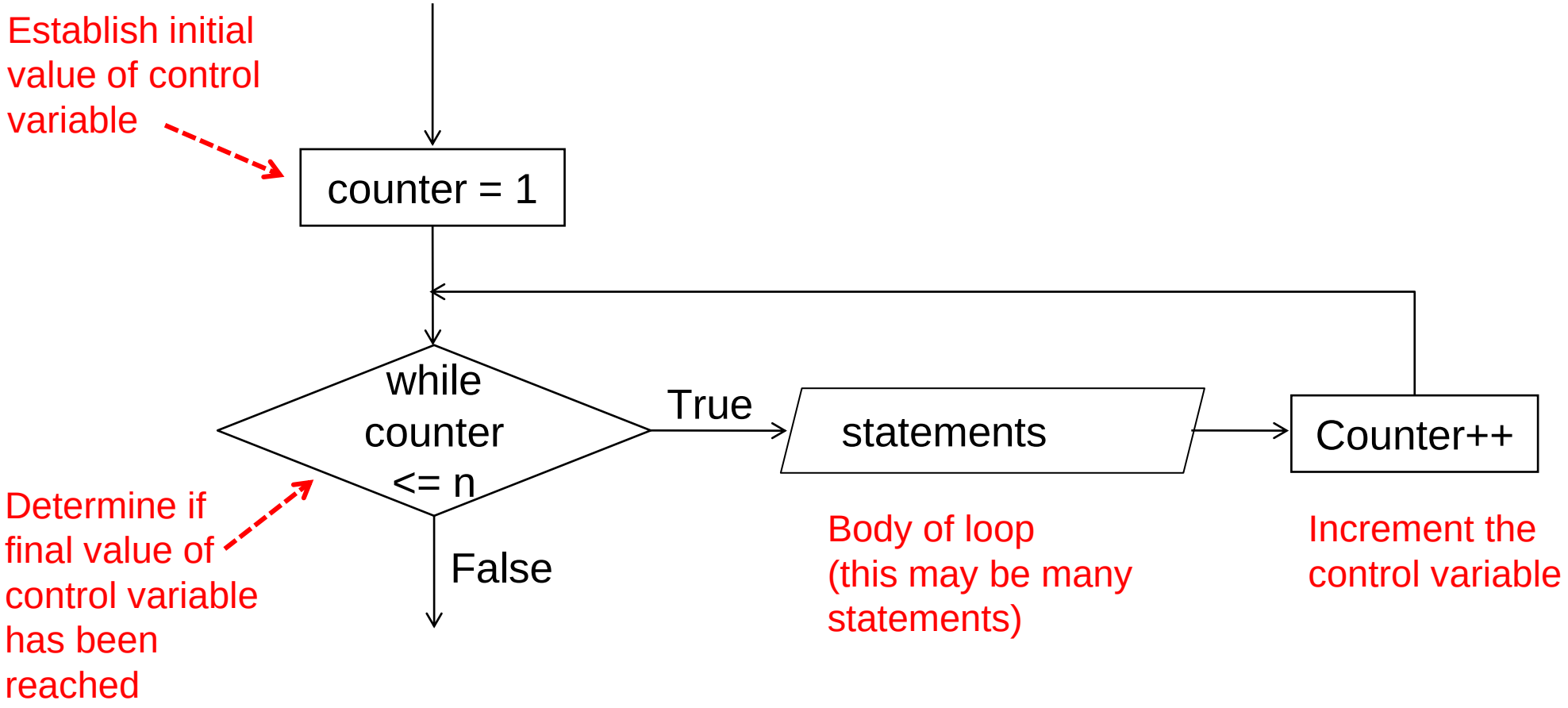
- when number of iterations is known

```
i = 1;  // INITIALIZATION
while (i <= n) {  // CONDITION
    statements
    i++;  // increment the counter
}
```

```
int counter = 1;              // initialization
while ( counter <= 10 )       // repetition condition
{
   printf( "%d\n", counter );
   counter++;                 // increment
}
```

Counter Controlled Loops

- The following flow chart illustrates the **while loop**.



Common Mistake

- forgetting to update the variable

```
i = 1;   // INITIALIZATION
while (i <= n) {   // CONDITION
    BLOCK
}
```

for Loops

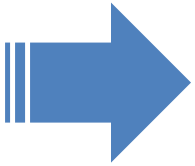
- for: combine initialization, condition, and update

```
for (INITIALIZATION; CONDITION; UPDATE) {
    Statements
}
```

- Expressions in the for statement are optional (they can be omitted.)
- But it is not recommended to omit them.

for Loop Example

```
i = 1;
while (i <= n) {
    printf("%d\n", i);
    i++;
}
```



```
for (i = 1; i <= n; i++){
    printf("%d\n", i);
}
```

Average Value Code

```
int i = 0;

age_total = 0;
for (i = 1; i <= n; i++) {
   printf("Age: ");
   scanf("%d", &age);
   age_total += age;
}
age_avg = (double) age_total / n;
```

Loop Variable

- loop variable can be defined at initialization
- can be used only within the loop

Loop Variable Example

```
int i = 0;

for (i = 1; i <= 10; i++) {
    ...
}
printf("%d\n", i);  // prints 11
```

```
for (int i = 1; i <= 10; i++) {
    ...
}
printf("%d\n", i);  // error: i is undefined
```

Closed Ranges

- using <= in condition creates a closed range

```
for (i = a; i <= b; i++)
```

- range is $[a..b]$
- includes a and b
- repeated $b - a + 1$ times

Half-Open Ranges

- using < in condition creates a half-open range

```
for (i = a; i < b; i++)
```

- range is $[a..b)$
- includes a, excludes b

- repeated $b - a$ times
- if $a = 0$, repeat b times
  - Very common in practice

Half-Open Range Example

```
age_total = 0;
for (int i = 0; i < n; i++) {
   printf("Age: ");
   scanf("%d", &age);
   age_total += age;
}
age_avg = (double) age_total / n;
```

Off-by-one Errors

- repeating once more or less than intended: off-by-one error
- very common problem

Count Examples

- what's the output?

```
for (i = 0; i <= 5; i++) {
    printf("%d\n", i);
}
```

Count Examples

- what's the output?

```
for (i = 5; i >= 0; i--) {
   printf("%d\n", i);
}
```

Count Examples

- what's the output?

```
for (i = 5; i > 0; i--) {
    printf("%d\n", i);
}
```

Count Examples

- what's the output?

```
for (i = 5; i > 0; i++) {
    printf("%d\n", i);
}
```

Count Examples

- what's the output?

```
for (i = 5; i > 5; i--) {
    printf("%d\n", i);
}
```

Count Examples

- what's the output?

```
for (i = 5; i < 5; i--) {
    printf("%d\n", i);
}
```

Count Examples

- what's the output?

```
for (i = 0; i < 9; i += 2) {
    printf("%d\n", i);
}
```

Count Examples

- what's the output?

```
for (i = 0; i != 9; i += 2) {
    printf("%d\n", i);
}
```

Count Examples

- what's the output?

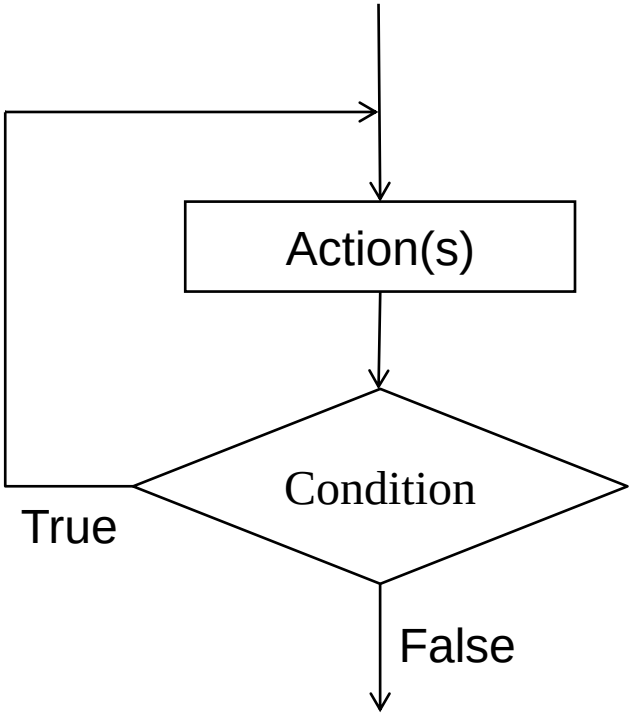
```
for (i = 1; i <= 20; i *= 2) {
    printf("%d\n", i);
}
```

do-while Loops

- test condition after executing block

```
do {
    BLOCK
} while (CONDITION);
```

- less common iterative statement



do-while Loops: Use Case

- possible use case: input validation
- check has to come after input is completed

```
do {
    printf("Enter age (between 18 and 65): ");
    scanf("%d", &age);
} while (age  < 18  || age  > 65);
```

Breaking the Loop

- some programmers prefer handling loop conditions inside loop body.

- create an infinite loop,

  get out of the loop when a condition is met

- condition to stop, not condition to continue

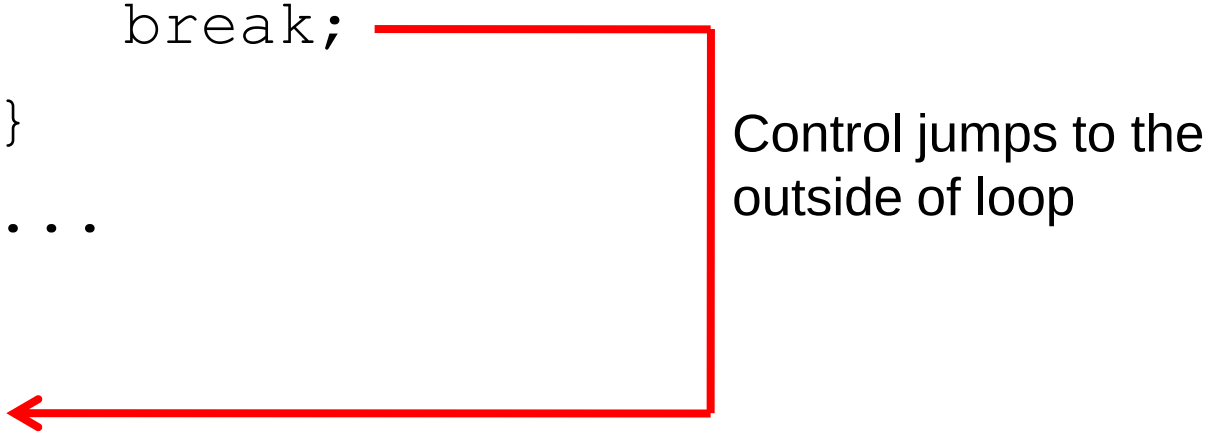
break Statement

- `break`
  - Causes immediate exit from a `while`, `for`, `do…while` or `switch` statement
  - Program execution continues with the first statement after the iteration or switch block that contains break
  - Common uses of the `break` statement
    - Escape early from a loop
    - Skip the remainder of a `switch` statement

break Statement

- get out of the **loop** that contains break

```
while(true)     {
    ...
    if (CONDITION)    {
        break;
    }
    ...
}
...
```



Example: break Statement

```
i = 1;     // Initialize loop counter
fact = 1; // Initialize factorial

while (true) {
    fact = fact * i;
    i = i + 1;
    if (i > n) {
        break;
    }
}
printf("Factorial : %d \n", fact);
```

Control jumps to the outside of loop

Example: break Statement

```c
/* Using the break statement in a for statement */
#include <stdio.h>
int main()
{
   int x; // counter

   // loop 10 times
   for ( x = 1; x <= 10; x++ )
   {
      // if x is 5, terminate loop
      if ( x == 5 ) {
         break; // break loop (exit) only if x is 5
      } // end if

      printf( "%d ", x ); // display value of x
   } // end for

   printf( "\nBroke out of loop at x == %d\n", x );
   return 0;
}
```

Control jumps to the outside of loop

Program Output

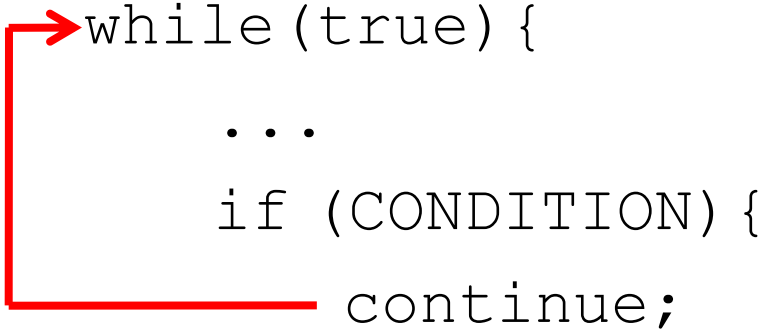
```
1 2 3 4
Broke out of loop at x == 5
```

continue Statement

- `continue`
  - Skips the remaining statements in the body of a while, for or do…while statement that contains continue 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

continue Statement

Control goes back to the next iteration of loop



```
while(true){
    ...
    if (CONDITION){
        continue;
    }
    ...
}
...
```

Example: continue Statement

```
i = 1;     // Initialize loop counter
fact = 1; // Initialize factorial

while (true) {
     fact = fact * i;
     i = i + 1;
     if (i <= n) {
          continue;
     }
     break;
}
printf("Factorial : %d \n", fact);
```

Control goes back to the next iteration of loop

Control jumps to the outside of loop

Example: continue Statement

Control goes back to next iteration of loop

x is incremented

```c
/* Using the continue statement in a for statement */
#include <stdio.h>
int main()
{
   int x; // counter

   // loop 10 times
   for ( 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
      } // end if

      printf( "%d ", x ); // display value of x
   } // end for

   printf( "\nUsed continue to skip printing the value 5\n" );
   return 0;
}
```

Program Output

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

Nested Loops

- Loops can be *nested* (i.e. loop in another loop).
- Indentation is recommended to clarify the loop structure.
- Example: The outer loop is controlled by **i** counter, the inner loop is controlled by **j** counter.
- **j** changes more frequently.

```
#include <stdio.h>
int main()
{
  int i, j; //Loop counters
  printf("I  J  10*I*J \n");
  printf("============ \n");
  for (i = 1; i <= 4; i++) { //outer loop
     for (j = 1; j <= 3; j++) { //inner loop
        printf("%d  %d  %d \n", i, j, 10*i*j);
     }//end of inner loop
     printf("--------- \n"); //separator line
  } //end of outer loop
  return 0;
} //end of main
```

Screen Output

```
I  J  10*I*J
============
1  1  10
1  2  20
1  3  30
---------
2  1  20
2  2  40
2  3  60
---------
3  1  30
3  2  60
3  3  90
---------
4  1  40
4  2  80
4  3  120
---------
```

Example: Left Pyramid

- Program displays a left-aligned pyramid with **nested** loops.

```
#include <stdio.h>
int main()
{
    int i, j;   //Loop counters
    int N = 10; //Define number of rows
    for (i = 1; i <= N; i++) { //Control the rows
        for (j = 1; j <= i; j++) {
            printf("*"); //Control the columns
        }
        printf("\n"); //Newline after each row
    }
}
```

Screen Output

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

Example: Right Pyramid

```c
#include <stdio.h>
int main()
{
  int i, j;
  int N = 10;

  for (i = 1; i <= N; i++)
  {
     for (j = 1; j <= N-i; j++)
         printf(" ");

     for (j = 1; j <= i; j++)
         printf("*");

     printf("\n");
  }

  return 0;
}
```

Screen Output

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

Extra Self Study Examples

Example: Determining a Prime Number

- The program below determines whether a given positive integer number is prime or not.
- If a number can not be divided by any other number, except 1 and itself, it is considered as a prime number. **(Prime numbers : 2, 3, 5, 7, 11, 13, 17, 19, 23, ….)**
- The program tries to divide the given number with all sequential numbers 2,3,4,5,6,7,8,9,10, ….. , up to one less than itself.

Program Output 1

```
Enter an integer number :23

23 is a prime number
```

Program Output 2

```
Enter an integer number :45

45  can be divided by 3 without a remainder

45 is NOT a prime number
```

```c
#include <stdio.h>
// Defining two constant symbols for logical comparisons.
#define TRUE 1  // we could use bool type from stdbool.h library as well.
#define FALSE 0

int main() {
    int x;    // user entered number
    int flag; // logical variable
    int i;    // divisor counter

    printf("Enter an integer number :"); scanf("%d", &x);

    flag = TRUE; // initial assumption (x is a prime)

    for (i=2; i < x; i++)
    {
       if (x % i == 0) // Check the divisibility of x by i
       {
         printf("\n %d  can be divided by %d without a remainder \n ", x, i);
         flag = FALSE; // x is not a prime (because x is divisible by i).
         break; // exit from loop
       }
    } // end for

    if (flag == TRUE)
       printf("%d is a prime number \n", x); // x is undivisible.
    else
       printf("%d is NOT a prime number \n", x); // x is divisible.
    return 0;
}//end of main
```

Instead of **i < x** condition, **i < x/2** can make the loop faster.

Example: Counting Frequencies of Letter Grades

- Program calculates the total counts (frequencies) of **letter grades** entered from keyboard.

```
// fig04_07.c
#include <stdio.h>
int main() {
   char grade;      // one letter grade
   int aCount = 0; // number of As
   int bCount = 0; // number of Bs
   int cCount = 0; // number of Cs
   int dCount = 0; // number of Ds
   int fCount = 0; // number of Fs

   printf(  "Enter the letter grades.\n"  );
   printf(  "Enter 0 to end input.\n"  );

   // Endless loop until user enters 0 as sentinel
   while ( 1 ) {
       grade = getchar(); // Read one char from keyboard
       if (grade == '0') break; // exit from loop
```

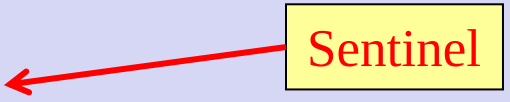
Part 1 of 3

```
// determine which grade was input
      switch ( grade ) { // switch nested in while
         case 'A': // grade was uppercase A
         case 'a': // or lowercase a
            ++aCount; // increment aCount
            break; // necessary to exit switch

         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; // optional; will exit switch anyway

      } // end of switch

   } // end of while


   // output summary of results
   printf( "\n Totals for each letter grade are: \n" );

   printf( "A: %d \n", aCount ); // display number of A grades
   printf( "B: %d \n", bCount ); // display number of B grades
   printf( "C: %d \n", cCount ); // display number of C grades
   printf( "D: %d \n", dCount ); // display number of D grades
   printf( "F: %d \n", fCount ); // display number of F grades

   return 0;
} // end of main
```

Program Output

```
Enter the letter grades.
Enter 0 to end input.
a
b
c
C
A
d
f
C
E
Incorrect letter grade entered. Enter a new grade.
D
A
b
0

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



Example: Sum of Even Numbers

- Program calculates the sum of even numbers from 2 to N.

```
/* Summation with the for loop */
#include <stdio.h>
int main()
{
   int sum = 0; // initialize sum
   int number;  // number to be added to sum

   for ( number = 2; number <= 100; number += 2 )
   {
      sum += number; // add number to sum
   }

   printf( "Sum is %d\n", sum ); // output sum
   return 0;
}
```

Sum = 2+4+6+8+ . . . +100 = 2550

Program Output

```
Sum is 2550
```

Example: Computing Taylor Series

Write a C program to calculate and display the cos(x) value, by using the following Taylor series.

$$\sum_{k=0}^{N} \frac{(-1)^k x^{2k}}{(2k)!}$$

Program should get the followings from user.
N : number of terms
X : degree

k is the looping counter.

The purpose is to calculate cos(x), without using the built-in cos(x) function.

Example: Computing Taylor Series

```c
#include <stdio.h>
#include <math.h> // for pow

int main() {
    double k, N, t, factorial;
    double X, fsum=0;

    printf("Enter N and X : ");
    scanf("%lf %lf", &N, &X);

    X = X*3.14 / 180; // Convert degree to radian.
    for (k = 0; k <= N; k++) {
        faktoriyel = 1;
        for (t = 1; t <= 2 * k; t++)
            factorial = factorial * t;
        fsum = fsum + pow(-1, k) * pow(X, 2 * k) / factorial;
    }
    printf("Result = %f \n", fsum);
    return 0;
}
```

Screen Output

```
Enter N and X : 50    30
Result = 0.866158
```

Square Root: Algorithm

1. g = 1
2. if $|g^2 - x| < 10^{-3}$ then g is the result, stop.
3. $g' = \frac{g + \frac{x}{g}}{2}$
4. replace g with g′ and go to step 2

Square Root: Variables

```
double x = 0.0; double
       guess = 0.0;
double improved_guess = 0.0;

printf("Number: ");
scanf("%lf", &x);
```

Square Root: Iteration

- what should we do at every iteration?
- improve the guess

```
improved_guess = (guess + (x / guess)) / 2;
guess = improved_guess;
```

Square Root: Stopping

- when should we stop?
- when the guess is close enough

```
#define TOLERANCE 1e-3


fabs(guess * guess - x) < TOLERANCE
```

- how long should we repeat?
- as long as the guess is not good enough

Square Root: Loop

- iteration:

```
while (fabs(guess *  guess  - x)  >= TOLERANCE){
    improved_guess = (guess + (x / guess)) / 2;
    guess = improved_guess;
}
```

- we have to set the initial guess before the loop

Square Root: Initial Value

```
guess = 1.0;  // initial guess
while (fabs(guess *  guess  - x)  >= TOLERANCE){
    improved_guess  = (guess  +  (x / guess)) / 2;
    guess = improved_guess;
}
```

Square Root: Calculation

- improved_guess variable is not necessary

```
guess = 1.0;  // initial guess
while (fabs(guess  * guess  -  x) >= TOLERANCE) {
    guess = (guess + (x / guess)) / 2;
}
```

Square Root Code

```
#include <stdio.h>  // printf, scanf
#include <math.h>   // fabs

#define TOLERANCE 1e-3

int main() {
    double x = 0.0;
    double guess = 0.0;

    printf("Number: ");
    scanf("%lf", &x);

    guess = 1.0;  // initial guess
    while (fabs(guess * guess - x) >= TOLERANCE) {
        guess = (guess + (x / guess)) / 2;
    }

    printf("%f\n", guess);
    return 0;
}
```

Skipping Loop

- if the condition is false to begin with,

  the block will not be executed at all

- example: incorrect condition

```
guess = 1.0;  // initial guess
while (fabs(guess *guess  - x) < TOLERANCE) {
    guess = (guess + (x / guess)) / 2;
}
```

Infinite Loop

- we have to make sure that the condition will eventually be false
- otherwise: infinite loop

- example: set tolerance to `1e-30`

Alternative Algorithm

- stop when guess cannot be improved anymore
- improved guess is very close to the current guess

- when should we continue?

```
fabs(improved_guess - guess) >= TOLERANCE
```

Square Root Alternative: Loop

```
guess = 1.0;  // initial guess
while (fabs(improved_guess - guess) >= TOLERANCE) {
    improved_guess = (guess + (x / guess)) / 2;
    guess = improved_guess;
}
```

- what should be the initial value for `improved_guess`? some value to
- make sure that we enter the loop

Square Root Alternative: Initial Values

```
guess = 1.0;  // initial guess
improved_guess = guess + 2 * TOLERANCE;
while (fabs(improved_guess - guess) >= TOLERANCE) {
    improved_guess = (guess + (x / guess)) / 2;
    guess = improved_guess;
}
```

- after the first iteration, at the time of check

  improved_guess - guess is always 0

Square Root Alternative: Fix

- replace current guess only if there is an improvement

```
guess = 1.0;  // initial guess
improved_guess = guess + 2 * TOLERANCE;
while (fabs(improved_guess -  guess)>= TOLERANCE) {
    improved_guess = (guess + (x / guess)) / 2;
    if (fabs(improved_guess - guess) >= TOLERANCE)   {
        guess = improved_guess;
    }
}
```

- checking the same condition twice

Square Root Alternative: Flag

```
guess = 1.0;   // initial guess
bool improving = true;
while (improving) {
    improved_guess =(guess + (x / guess)) / 2;
    if (fabs(improved_guess - guess) >= TOLERANCE) {
        guess = improved_guess;
    } else {
        improving = false;
    }
}
```

Square Root Code: Break

```
guess = 1.0;  // initial guess
while (true) {
    improved_guess =(guess + (x  / guess)) / 2;
    if (fabs(improved_guess - guess) < TOLERANCE)  {
        break;
    }
    guess = improved_guess;
}
```