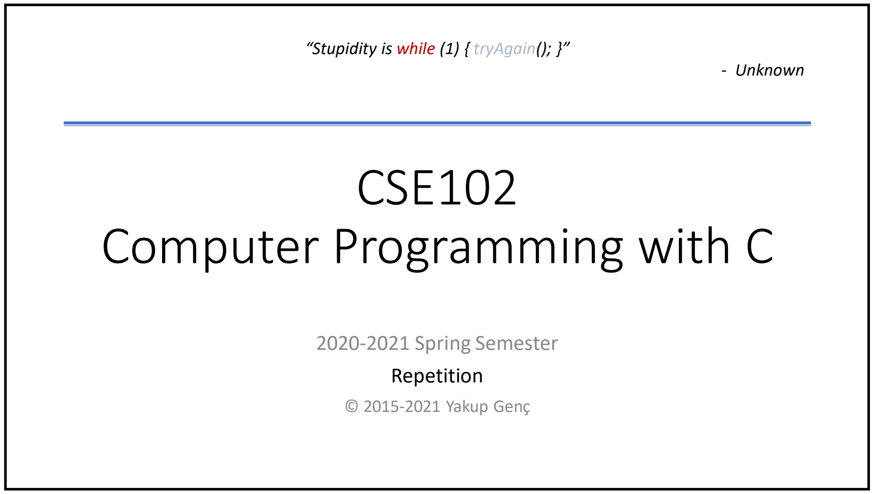


"Stupidity is while (1) { tryAgain(); }"

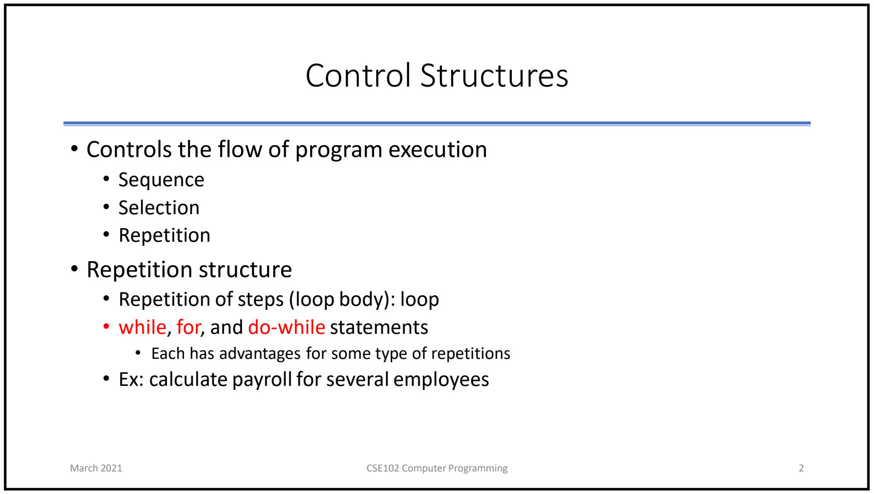
- Unknown

CSE102
Computer Programming with C

2020-2021 Spring Semester

Repetition

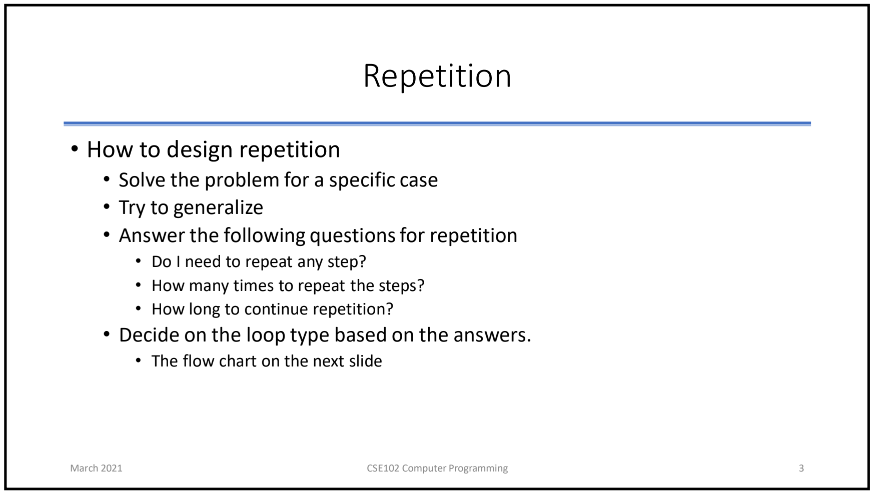
1



Control Structures

- Controls the flow of program execution
  - Sequence
  - Selection
  - Repetition
- Repetition structure
  - Repetition of steps (loop body): loop
  - while, for, and do-while statements
    - Each has advantages for some type of repetitions
  - Ex: calculate payroll for several employees

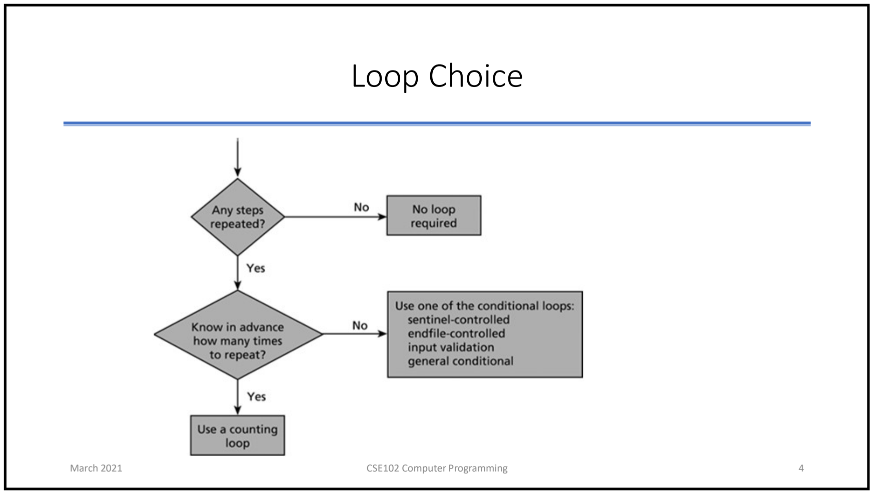
2



Repetition

- How to design repetition
  - Solve the problem for a specific case
  - Try to generalize
  - Answer the following questions for repetition
    - Do I need to repeat any step?
    - How many times to repeat the steps?
    - How long to continue repetition?
  - Decide on the loop type based on the answers.
    - The flow chart on the next slide

3



Loop Choice

4

Comparison of Loop Kinds

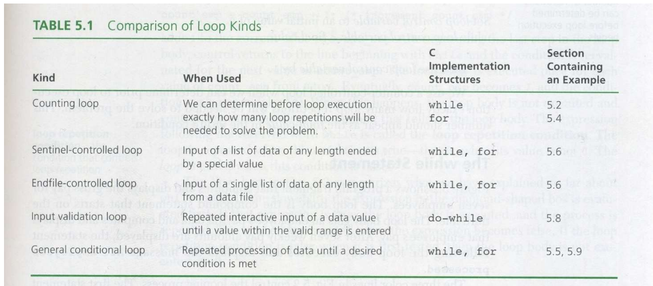


TABLE 5.1 Comparison of Loop Kinds

Kind	When Used	C Implementation Structures	Section Containing an Example
Counting loop	We can determine before loop execution exactly how many loop repetitions will be needed to solve the problem.	`while` `for`	5.2 5.4
Sentinel-controlled loop	Input of a list of data of any length ended by a special value	`while, for`	5.6
Endfile-controlled loop	Input of a single list of data of any length from a data file	`while, for`	5.6
Input validation loop	Repeated interactive input of a data value until a value within the valid range is entered	`do-while`	5.8
General conditional loop	Repeated processing of data until a desired condition is met	`while, for`	5.5, 5.9

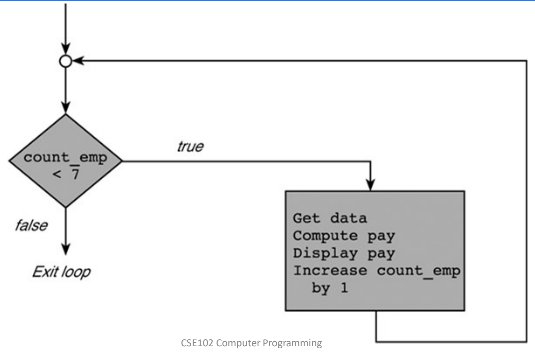
Counter Controlled Loop

- Repetition is managed by a loop control variable
  - For example a counter

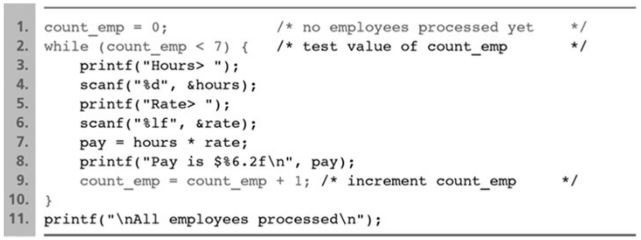
General format:

```
set counter to 0                /* initialization */
while counter < final value     /* test           */
      do one or more things     /* loop body      */
      increase counter by one   /* updating       */
```

Flowchart for a while Loop



Program Fragment with a Loop



```
count_emp = 0;              /* no employees processed yet    */
while (count_emp < 7) {     /* test value of count_emp       */
    printf("Hours> ");
    scanf("%d", &hours);
    printf("Rate> ");
    scanf("%lf", &rate);
    pay = hours * rate;
    printf("Pay is $%6.2f\n", pay);
    count_emp = count_emp + 1; /* increment count_emp     */
}
printf("\nAll employees processed\n");
```

while statement

General syntax:

```
while (loop repetition control)
  statement
```

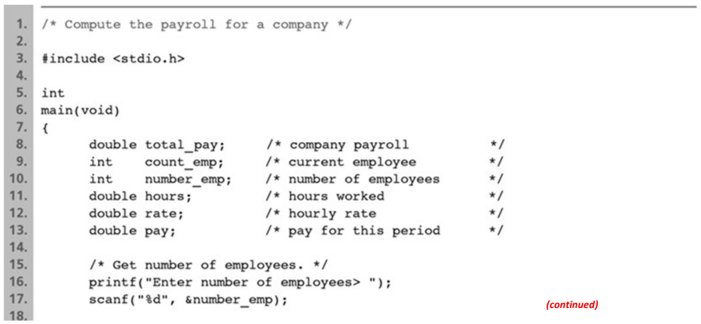
Example

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

Payroll calculator

- Calculate payroll for several employees
  - Calculate the total payroll as well
- Input:
  - For each employee
    - Hours, rate, pay
  - Number of employees
- Output
  - For each employee
    - Payroll
  - Total payroll

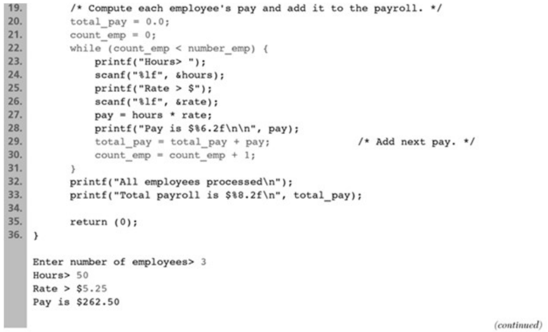
Payroll calculator



```
/* 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);

```

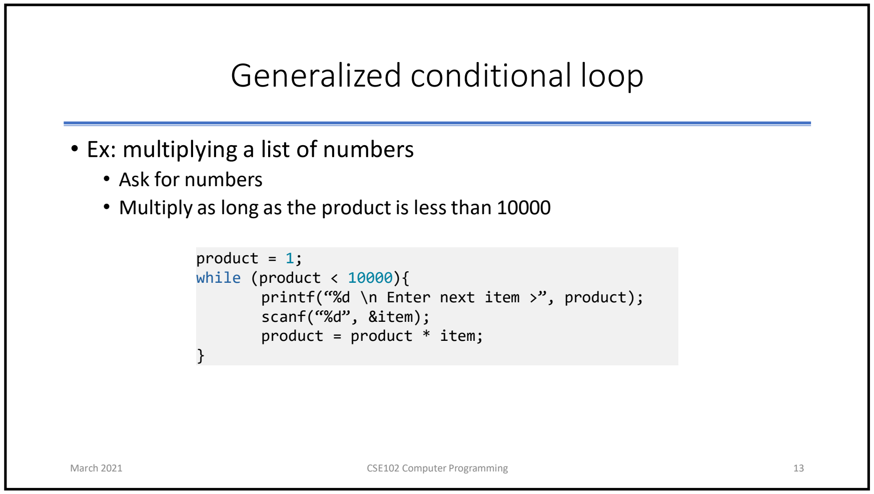
(continued)

Payroll calculator



```
      /* 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
```

(continued)

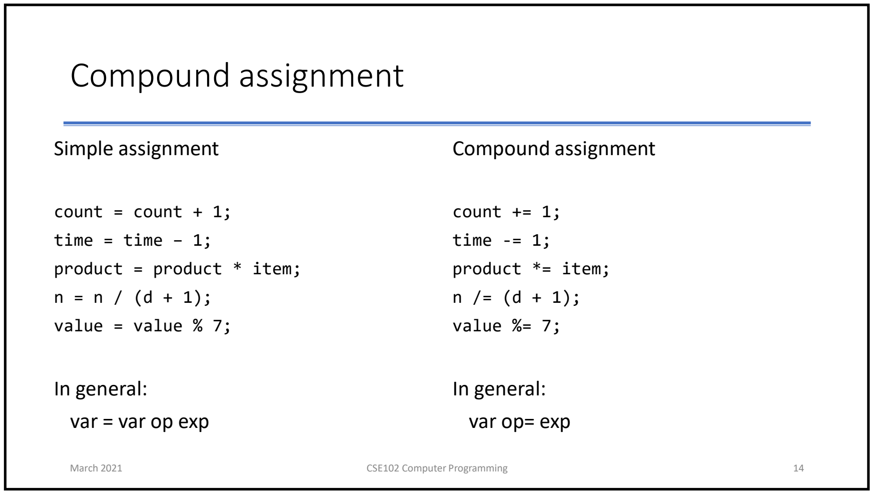


Generalized conditional loop

- Ex: multiplying a list of numbers
  - Ask for numbers
  - Multiply as long as the product is less than 10000

```
product = 1;
while (product < 10000){
      printf(“%d \n Enter next item >”, product);
      scanf(“%d”, &item);
      product = product * item;
}
```

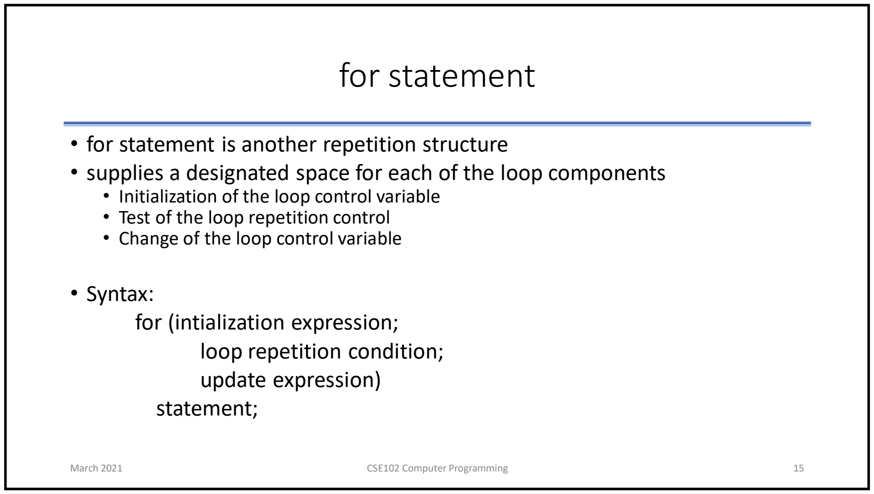
13



Compound assignment

Simple assignment	Compound assignment
`count = count + 1;`	`count += 1;`
`time = time – 1;`	`time -= 1;`
`product = product * item;`	`product *= item;`
`n = n / (d + 1);`	`n /= (d + 1);`
`value = value % 7;`	`value %= 7;`
In general: var = var op exp	In general: var op= exp

14

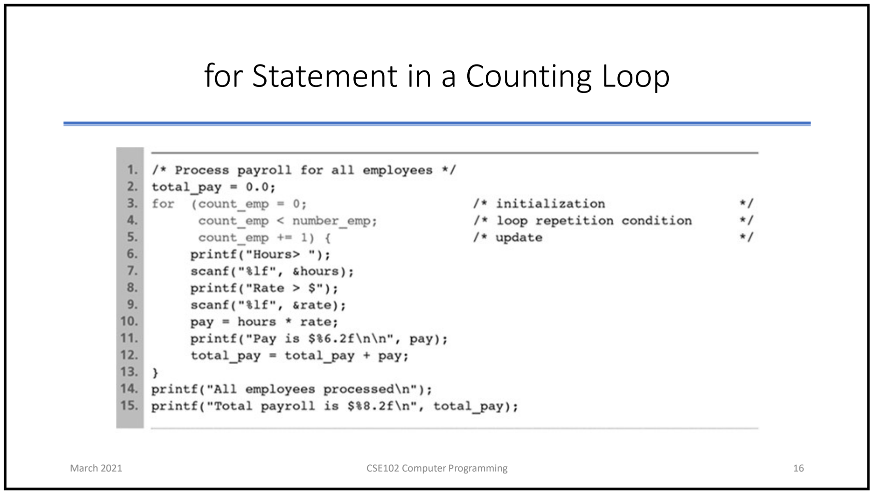


for statement

- for statement is another repetition structure
- supplies a designated space for each of the loop components
  - Initialization of the loop control variable
  - Test of the loop repetition control
  - Change of the loop control variable
- Syntax:

```
    for (intialization expression;
            loop repetition condition;
            update expression)
        statement;
```

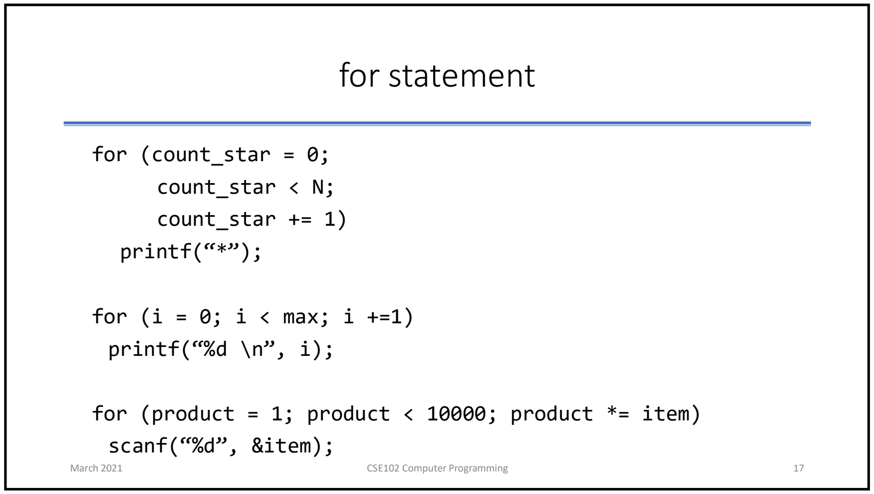
15



for Statement in a Counting Loop

```
/* Process payroll for all employees */
total_pay = 0.0;
for  (count_emp = 0;                    /* initialization             */
      count_emp < number_emp;           /* loop repetition condition  */
      count_emp += 1) {                 /* update                     */
     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;
}
printf("All employees processed\n");
printf("Total payroll is $%8.2f\n", total_pay);
```

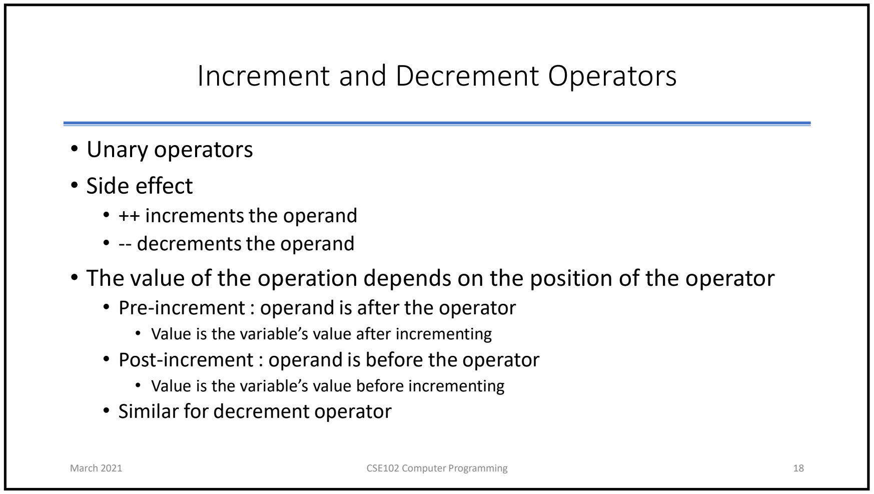
16



for statement

```
for (count_star = 0;
     count_star < N;
     count_star += 1)
  printf("*");

for (i = 0; i < max; i +=1)
 printf("%d \n", i);

for (product = 1; product < 10000; product *= item)
 scanf("%d", &item);
```

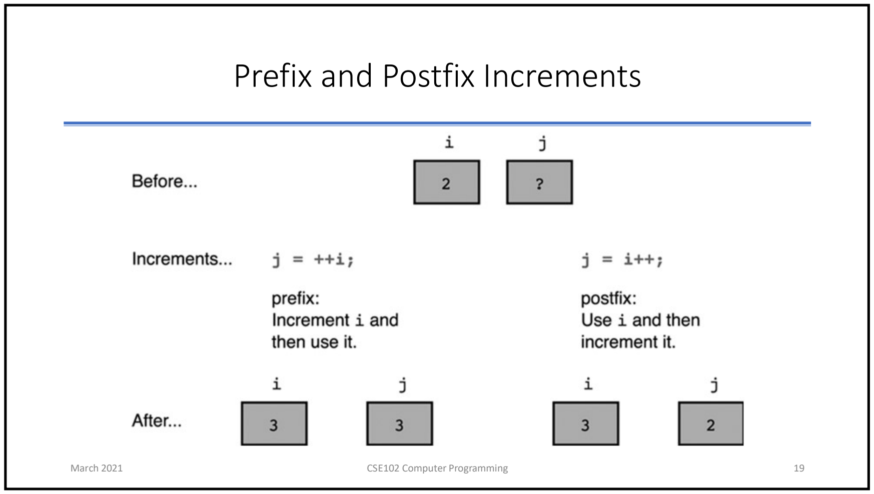
17



Increment and Decrement Operators

- Unary operators
- Side effect
  - ++ increments the operand
  - -- decrements the operand
- The value of the operation depends on the position of the operator
  - Pre-increment : operand is after the operator
    - Value is the variable's value after incrementing
  - Post-increment : operand is before the operator
    - Value is the variable's value before incrementing
  - Similar for decrement operator

18



Prefix and Postfix Increments

Before...

i j

2 ?

Increments...

`j = ++i;`

prefix:
Increment `i` and
then use it.

`j = i++;`

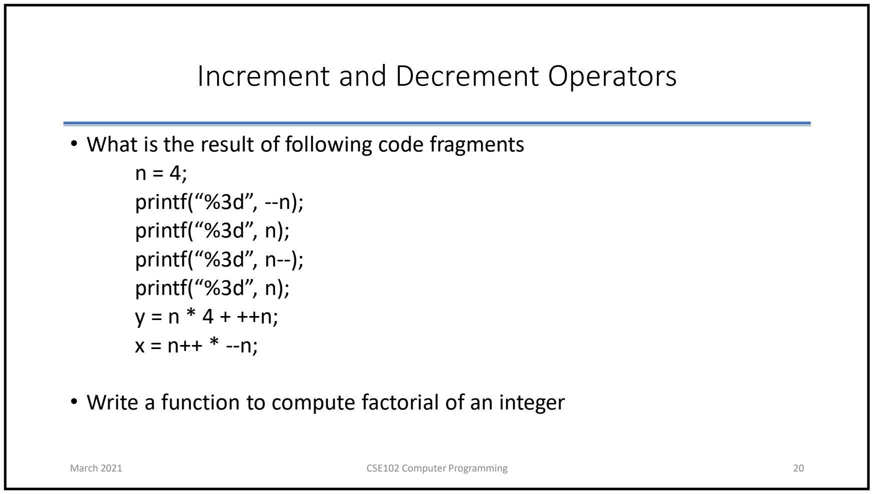
postfix:
Use `i` and then
increment it.

After...

i j i j

3 3 3 2

19



Increment and Decrement Operators

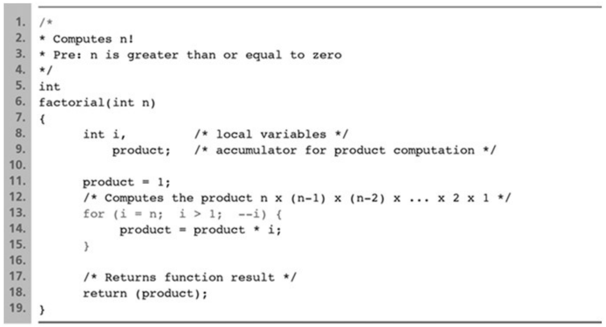
- What is the result of following code fragments

```
n = 4;
printf("%3d", --n);
printf("%3d", n);
printf("%3d", n--);
printf("%3d", n);
y = n * 4 + ++n;
x = n++ * --n;
```

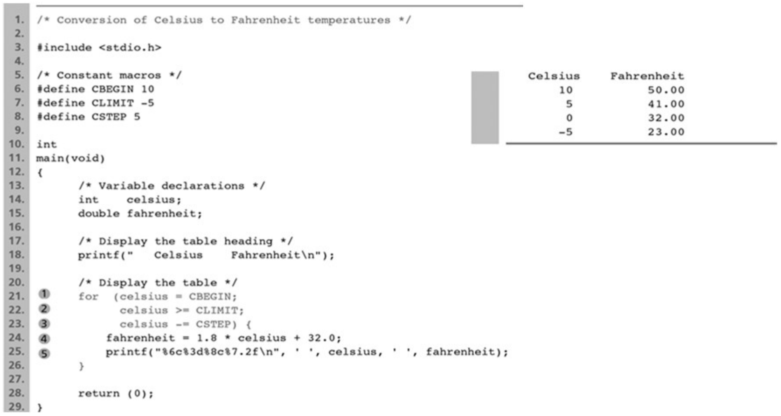
- Write a function to compute factorial of an integer

20

Function to Compute Factorial



```
/*
* Computes n!
* Pre: n is greater than or equal to zero
*/
int
factorial(int n)
{
      int i,          /* local variables */
          product;    /* accumulator for product computation */

      product = 1;
      /* Computes the product n x (n-1) x (n-2) x ... x 2 x 1 */
      for (i = n;  i > 1;  --i) {
           product = product * i;
      }

      /* Returns function result */
      return (product);
}
```



```
/* Conversion of Celsius to Fahrenheit temperatures */

#include <stdio.h>

/* Constant macros */
#define CBEGIN 10
#define CLIMIT -5
#define CSTEP 5

int
main(void)
{
      /* Variable declarations */
      int    celsius;
      double fahrenheit;

      /* Display the table heading */
      printf("   Celsius    Fahrenheit\n");

      /* Display the table */
①     for  (celsius = CBEGIN;
②           celsius >= CLIMIT;
③           celsius -= CSTEP) {
④          fahrenheit = 1.8 * celsius + 32.0;
⑤          printf("%6c%3d%8c%7.2f\n", ' ', celsius, ' ', fahrenheit);
      }

      return (0);
}
```

```
   Celsius    Fahrenheit
      10         50.00
       5         41.00
       0         32.00
      -5         23.00
```

Conditional Loops

- If you do not know exact number of repetitions
- Ex: ensuring valid user input
  - Continue to prompt user to enter a value as long as the response is not reasonable

```
Print an initial prompting message
Get the number of observed values
While the number of value is negative
  Print a warning message and ask for another value
  Get the number of observed values
```

- Where is initialization, test and update steps?
  - How to write the loop in C?

Conditional Loops

- Ex: Monitoring gasoline supply
  - Capacity 80000 barrels
  - Use of gasoline is entered in gallons
    - 1 barrel = 42 gallons
  - Alert if the supply falls below 10% of the capacity
- Input:
  - Current supply
  - Several uses
- Output
  - Remaining supply
  - Alert

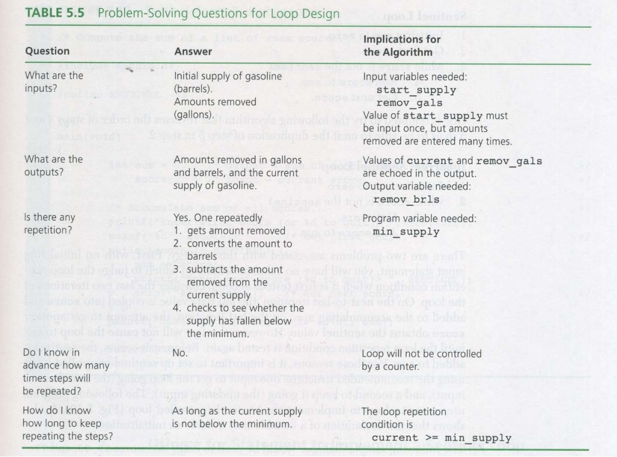


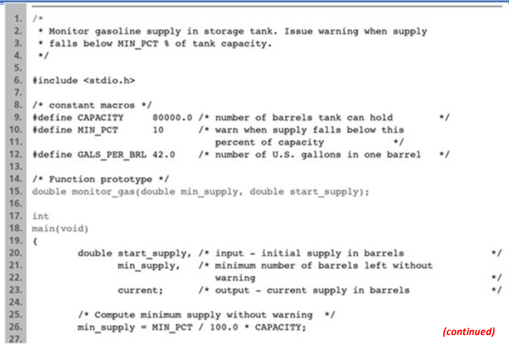
TABLE 5.5 Problem-Solving Questions for Loop Design

Question	Answer	Implications for the Algorithm

What are the inputs?	Initial supply of gasoline (barrels).
	Amounts removed (gallons).
		Input variables needed:
		`start_supply`
		`remov_gals`
		Value of `start_supply` must be input once, but amounts removed are entered many times.

What are the outputs?	Amounts removed in gallons and barrels, and the current supply of gasoline.	Values of `current` and `remov_gals` are echoed in the output. Output variable needed: `remov_brls`
Is there any repetition?	Yes. One repeatedly 1. gets amount removed 2. converts the amount to barrels 3. subtracts the amount removed from the current supply 4. checks to see whether the supply has fallen below the minimum.	Program variable needed: `min_supply`
Do I know in advance how many times steps will be repeated?	No.	Loop will not be controlled by a counter.
How do I know how long to keep repeating the steps?	As long as the current supply is not below the minimum.	The loop repetition condition is `current >= min_supply`

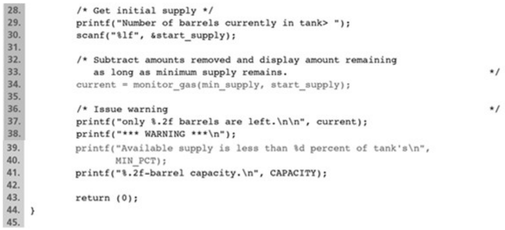
Monitoring Gasoline Storage Tank



```
/*
 * Monitor gasoline supply in storage tank. Issue warning when supply
 * falls below MIN_PCT % of tank capacity.
 */

#include <stdio.h>

/* constant macros */
#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;

```

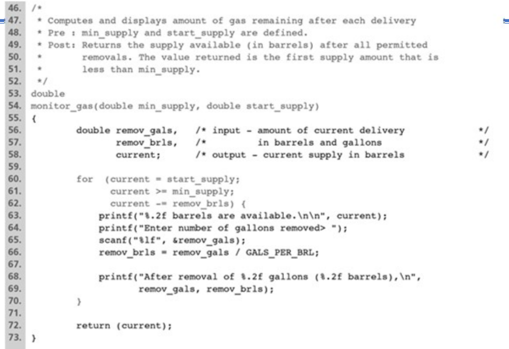
(continued)

Monitoring Gasoline Storage Tank

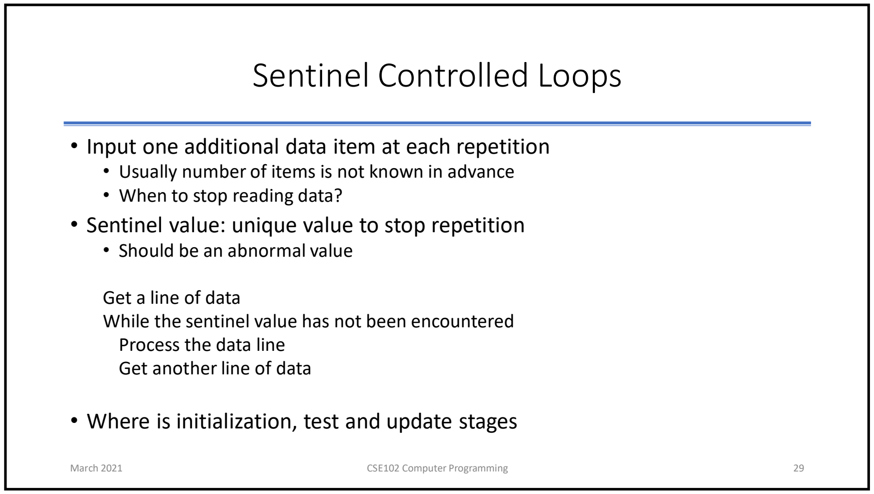


```
        /* Get initial supply */
        printf("Number of barrels currently in tank> ");
        scanf("%lf", &start_supply);

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

        /* Issue warning                                                        */
        printf("only %.2f barrels are left.\n\n", current);
        printf("*** WARNING ***\n");
        printf("Available supply is less than %d percent of tank's\n",
               MIN_PCT);
        printf("%.2f-barrel capacity.\n", CAPACITY);

        return (0);
}

```

Monitoring Gasoline Storage Tank



```
/*
 * Computes and displays amount of gas remaining after each delivery
 * Pre : min_supply and start_supply are defined.
 * Post: Returns the supply available (in barrels) after all permitted
 *       removals. The value returned is the first supply amount that is
 *       less than min_supply.
 */
double
monitor_gas(double min_supply, double start_supply)
{
        double remov_gals,   /* input - amount of current delivery           */
               remov_brls,   /*         in barrels and gallons               */
               current;      /* output - current supply in barrels           */

        for  (current = start_supply;
              current >= min_supply;
              current -= remov_brls) {
            printf("%.2f barrels are available.\n\n", current);
            printf("Enter number of gallons removed> ");
            scanf("%lf", &remov_gals);
            remov_brls = remov_gals / GALS_PER_BRL;

            printf("After removal of %.2f gallons (%.2f barrels),\n",
                   remov_gals, remov_brls);
        }

        return (current);
}
```

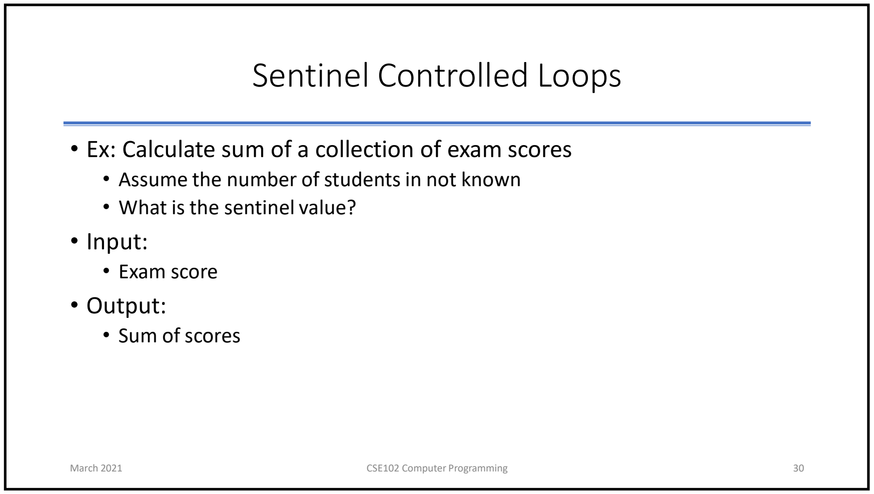


Sentinel Controlled Loops

- Input one additional data item at each repetition
  - Usually number of items is not known in advance
  - When to stop reading data?
- Sentinel value: unique value to stop repetition
  - Should be an abnormal value

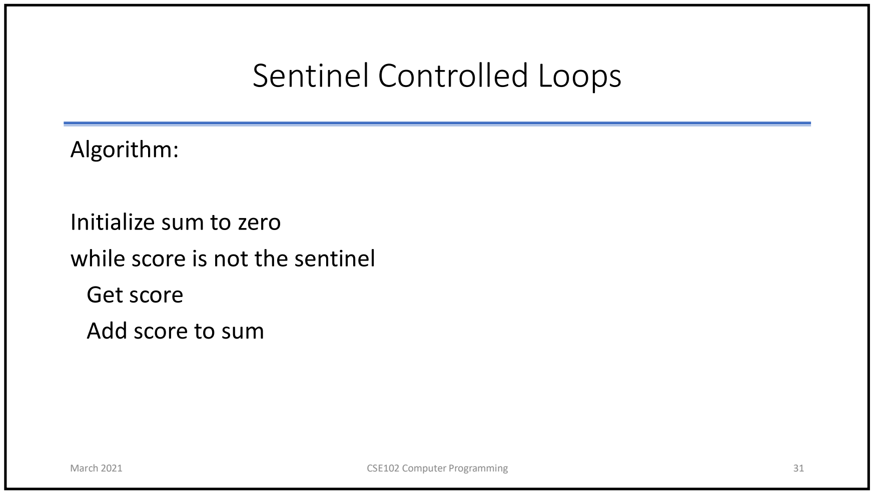
```
Get a line of data
While the sentinel value has not been encountered
  Process the data line
  Get another line of data
```

- Where is initialization, test and update stages



Sentinel Controlled Loops

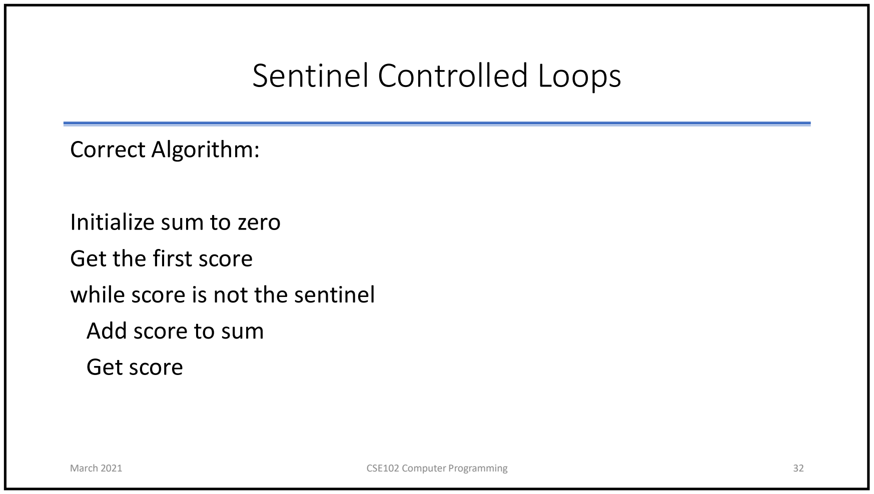
- Ex: Calculate sum of a collection of exam scores
  - Assume the number of students in not known
  - What is the sentinel value?
- Input:
  - Exam score
- Output:
  - Sum of scores



Sentinel Controlled Loops

Algorithm:

```
Initialize sum to zero
while score is not the sentinel
  Get score
  Add score to sum
```

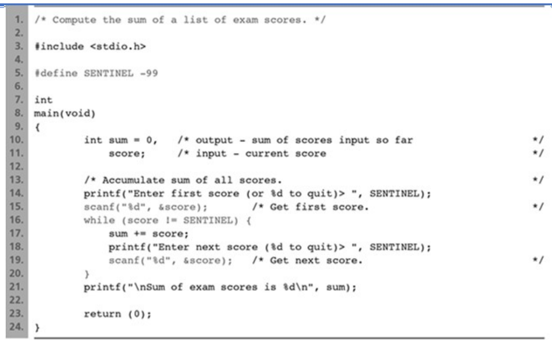


Sentinel Controlled Loops

Correct Algorithm:

```
Initialize sum to zero
Get the first score
while score is not the sentinel
  Add score to sum
  Get score
```

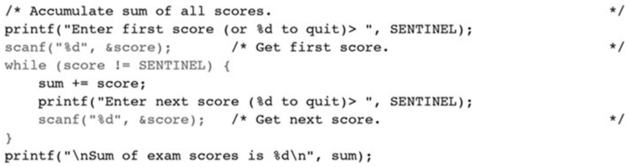
Sentinel-Controlled while Loop



```
/* Compute the sum of a list of exam scores. */

#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);
}
```

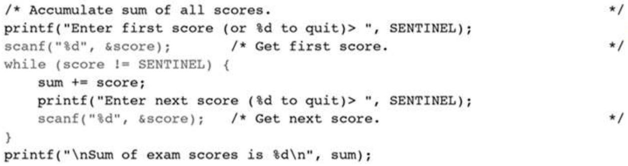
Sentinel-Controlled for Loop

- Can we use for statement for sentinel controlled loops?



```
/* 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);
```

Sentinel-Controlled for Loop



```
/* 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);
```

```
printf(……);
for (scanf(“%d”,&score);
        score != SENTINEL;
         scanf(“%d”,&score)) {
  sum += score;
  printf(…….);
}
```

End-file Controlled Loops

Ex: Calculate sum of a list of integers in a file

- A data file is terminated by an endfile character
    - detected by fscanf functions.
- special sentinel value is not required
    - uses the status value returned by fscanf

Algorithm:

```
Initialize sum to zero
Read the first value
while end of file is not reached
    Add value to sum
    Read the next value
```

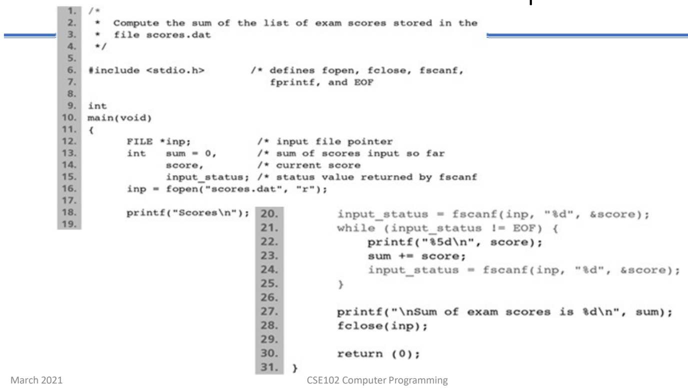
```
*******
*******
*******
*******
*******
*******
*******
```

```
int i,j,n;

n = 7;

for (j=0;j<n;j++) {
  for (i=0;i<n;i++) printf(“*”);
  printf(“\n”);
}
```

```
void printline(int n);
void main () {
  int i,j,n;
  n = 7;
  for (j=0;j<n;j++) {
     printline(n);
  }
}
```

```
void printline(int n) {
  int i;
  for (i=0;i<n;i++) rintf(“*”);
  printf(“\n”;
}
```

End-file Controlled Loops



```
/*
 *  Compute the sum of the list of exam scores stored in the
 *  file scores.dat
 */

#include <stdio.h>       /* defines fopen, fclose, fscanf,
                            fprintf, and EOF

int
main(void)
{
      FILE *inp;          /* input file pointer
      int   sum = 0,      /* sum of scores input so far
            score,        /* current score
            input_status; /* status value returned by fscanf
      inp = fopen("scores.dat", "r");

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

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

      return (0);
}
```

Infinite Loop on Faulty Data

- If the file contains a faulty data 7o, fscanf
  - stops at the letter ‘o’,
  - stores the value 7 in score
  - leaves the letter ‘o’ unprocessed.
  - returns a status value of one
- On the next loop iteration, fscanf
  - finds the letter ‘o’ awaiting processing
  - leaves the variable score unchanged
  - leaves the letter ' o ' unprocessed,
  - returns a status value of zero
- In the previous program
  - the return value of fscanf is not checked for values other than EOF
  - unsuccessful attempt to process the letter ‘o’ repeats over and over.

Infinite loop!...

Infinite Loop on Faulty Data

- Solution: Change the loop repetition condition to
  while (input_status == 1)
- loop exits on
  - end of file (input_status negative) OR
  - faulty data (input_status zero)
- Add an if statement after the loop to decide whether to print the results or to warn of bad input.

```
if (input_status == EOF)
   printf (’Sum of exam scores is %d\n", sum);
else {
      fscanf (inp, “%c”, &bad_char);
      printf("*** Error in input: %c ***\", bad_char);
}
```

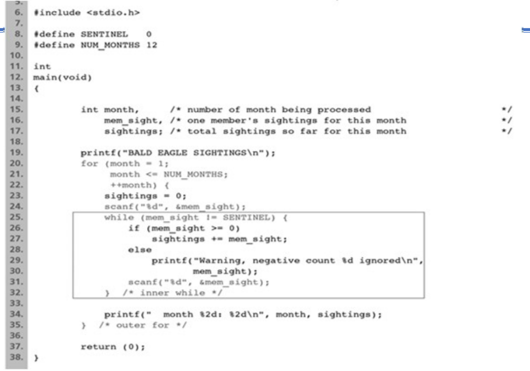
Nested Loops

- Loops may be nested like other control structures.
  - an outer loop with one or more inner loops.
  - Each time the outer loop is repeated, the inner loops are reentered,

Ex: Audubon Club members’ sightings of bald eagles

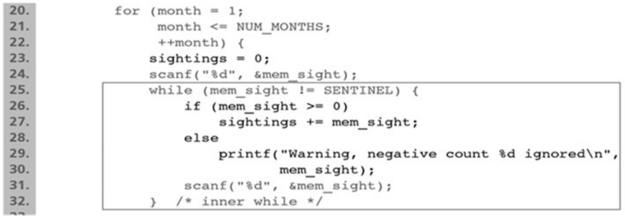
- Input: for each month a group of integers followed by a zero
- Output: for each month total sightings
- program contains a sentinel loop (for sightings in a month) nested within a counting loop (for months).

Nested Loops



```
#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);
}
```

Nested Loops

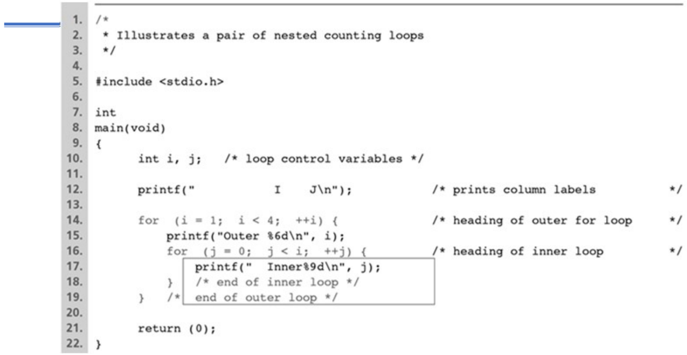


```
for (month = 1;
     month <= NUM_MONTHS;
     ++month) {
    sightings = 0;
    scanf("%d", &mem_sight);
    while (mem_sight != SENTINEL) {
        if (mem_sight >= 0)
            sightings += mem_sight;
        else
            printf("Warning, negative count %d ignored\n",
                  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

Nested Loops

- Ex: a simple program with two nested counting loops.
  - The outer loop is repeated three times (for i = 1, 2 3).
  - The number of times the inner loop is repeated depends on the current value of i.

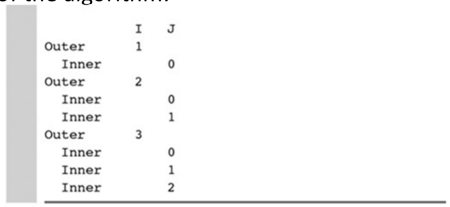
Nested Loops



```
/*
 * Illustrates a pair of nested counting loops
 */

#include <stdio.h>

int
main(void)
{
      int i, j;   /* loop control variables */

      printf("          I    J\n");           /* prints column labels         */

      for  (i = 1;  i < 4;  ++i) {            /* heading of outer for loop    */
          printf("Outer %6d\n", i);
          for  (j = 0;  j < i;  ++j) {        /* heading of inner loop        */
              printf("  Inner%9d\n", j);
          }  /* end of inner loop */
      }   /* end of outer loop */

      return (0);
}
```

Nested Loops

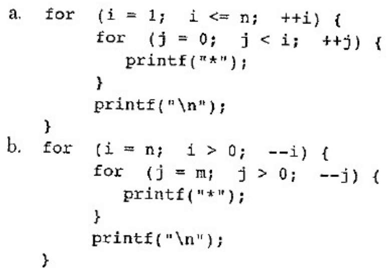
- The output of the algorithm:



```
             I   J
Outer        1
  Inner          0
Outer        2
  Inner          0
  Inner          1
Outer        3
  Inner          0
  Inner          1
  Inner          2
```

Nested Loops

- What is displayed by the following program segments, assuming m is 3 and n is 5?



```
a.  for  (i = 1;  i <= n;  ++i) {
          for  (j = 0;  j < i;  ++j) {
              printf("*");
          }
          printf("\n");
    }
b.  for  (i = n;  i > 0;  --i) {
          for  (j = m;  j > 0;  --j) {
              printf("*");
          }
          printf("\n");
    }
```

do-while Statement

- for statements and while statements evaluate loop repetition condition before the first execution of the loop body.
- Pretest is usually undesirable
  - when there may be no data items to process
  - when the initial value of the loop control variable is outside its expected range.
- Sometimes loop must execute at least once
- Ex: interactive input

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

do-while Statement

- C provides the do-while statement to implement such loops

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

```
do {
        printf("Enter a letter from A to E> ");
        scanf("%c", &letter);
} while (letter < 'A' || letter > 'E');
```

do-while Statement

- SYNTAX:

```
do {
   statements
} while ( loop repetition condition );
```

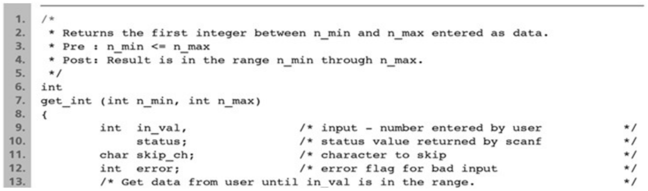
- Ex: Find first even input

```
do
        status = scanf("%d", &num);
while (status > 0  && (num % 2) != 0);
```

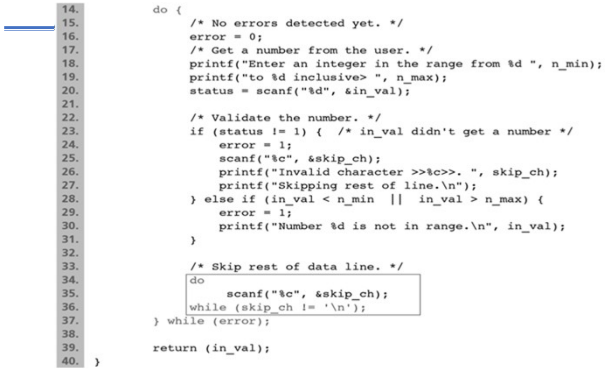
Flag Controled Loops

- If loop repetition condition is complex
  - Use a flag is a type int (values: **1** (true) and **0** (false))
  - Flag represents whether a certain event has occurred.

- Ex: Input Validation
  - The do-while is often used in checking for valid input
    - An input is always needed
  - Two nested loops
    - Repeat reading input when the input is not valid
      - not in range OR not a number
    - Repeat reading input to skip invalid input line
      - Not to have infinite loop



```
/*
 * Returns the first integer between n_min and n_max entered as data.
 * Pre : n_min <= n_max
 * Post: Result is in the range n_min through n_max.
 */
int
get_int (int n_min, int n_max)
{
        int  in_val,                /* input - number entered by user          */
             status;                /* status value returned by scanf          */
        char skip_ch;               /* character to skip                       */
        int  error;                 /* error flag for bad input                */
        /* Get data from user until in_val is in the range.                   */
```

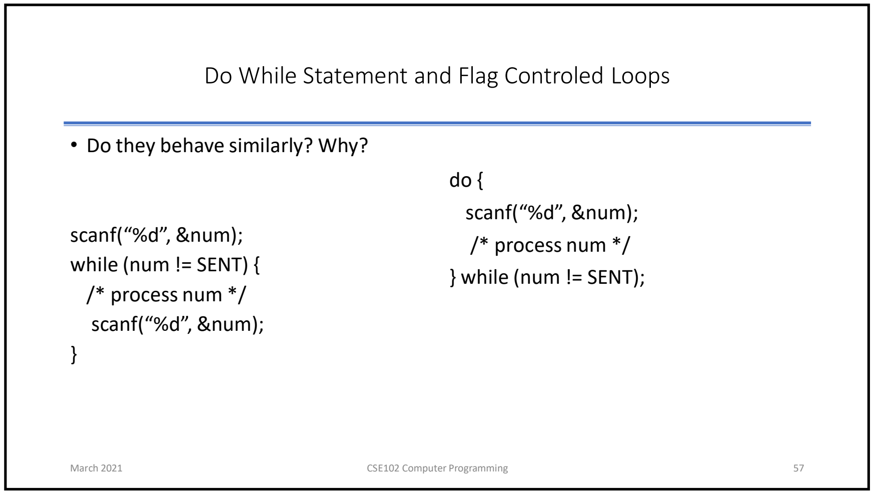


```
        do {
             /* No errors detected yet. */
             error = 0;
             /* Get a number from the user. */
             printf("Enter an integer in the range from %d ", n_min);
             printf("to %d inclusive> ", n_max);
             status = scanf("%d", &in_val);

             /* Validate the number. */
             if (status != 1) {  /* in_val didn't get a number */
                 error = 1;
                 scanf("%c", &skip_ch);
                 printf("Invalid character >>%c>>. ", skip_ch);
                 printf("Skipping rest of line.\n");
             } else if (in_val < n_min  ||  in_val > n_max) {
                 error = 1;
                 printf("Number %d is not in range.\n", in_val);
             }

             /* Skip rest of data line. */
             do
                  scanf("%c", &skip_ch);
             while (skip_ch != '\n');
        } while (error);

        return (in_val);
}
```

Flag Controled Loops

- Execution results

```
Enter an integer in the range from 10 to 20 inclusive> @20
Invalid character >>@>>. Skipping rest of line.
Enter an integer in the range from 10 to 20 inclusive> 2o
Number 2 is not in range.
Enter an integer in the range from 10 to 20 inclusive> 20
```

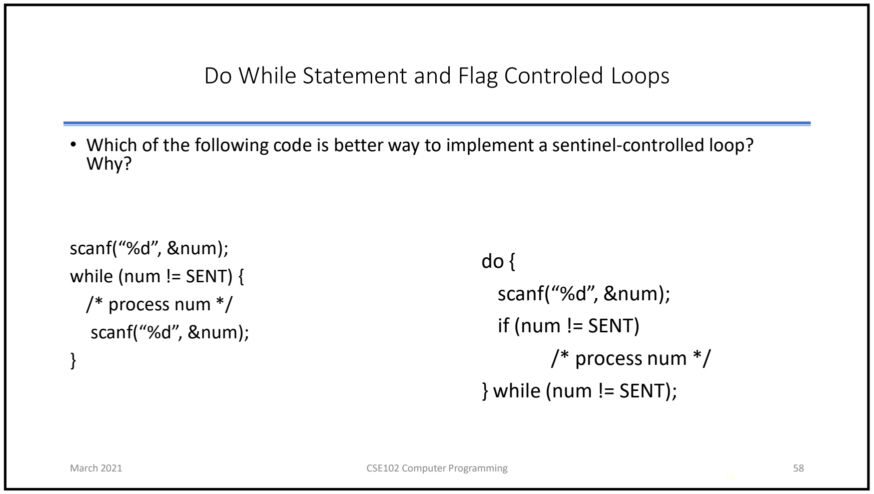


Do While Statement and Flag Controled Loops

- Do they behave similarly? Why?

```
scanf(“%d”, &num);
while (num != SENT) {
   /* process num */
    scanf(“%d”, &num);
}
```

```
do {
   scanf(“%d”, &num);
    /* process num */
} while (num != SENT);
```

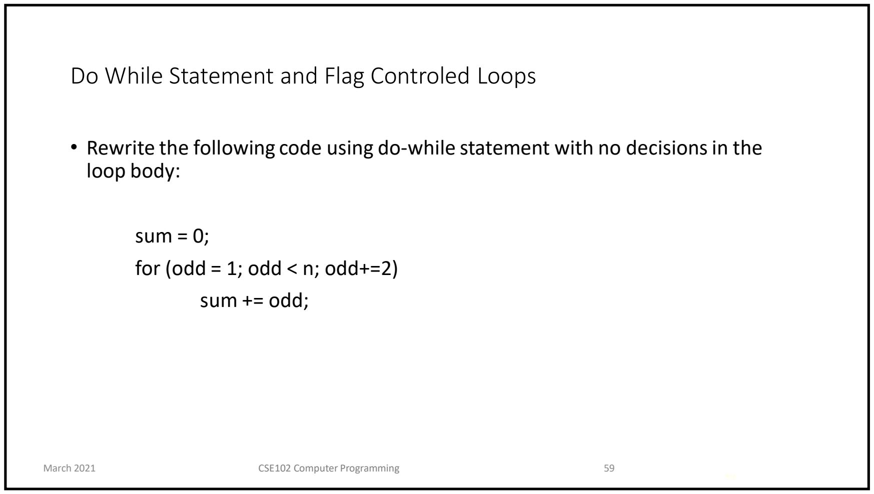


Do While Statement and Flag Controled Loops

- Which of the following code is better way to implement a sentinel-controlled loop? Why?

```
scanf(“%d”, &num);
while (num != SENT) {
   /* process num */
    scanf(“%d”, &num);
}
```

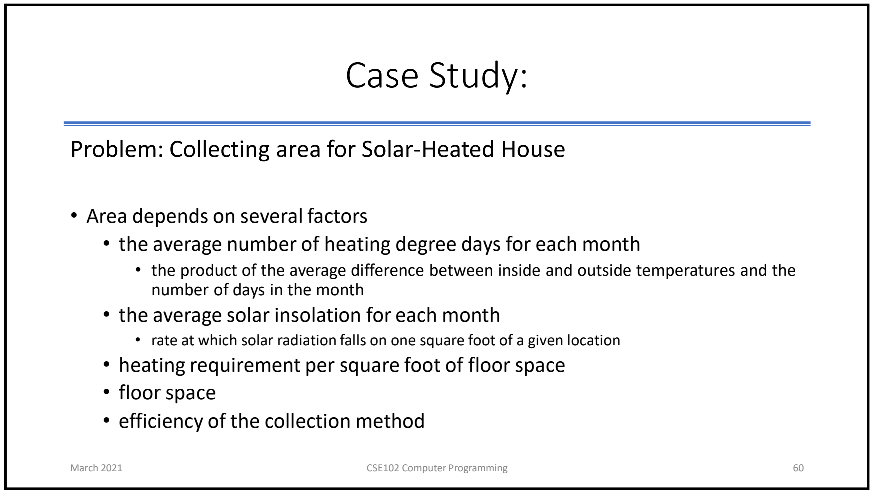
```
do {
   scanf(“%d”, &num);
   if (num != SENT)
            /* process num */
} while (num != SENT);
```



Do While Statement and Flag Controled Loops

- Rewrite the following code using do-while statement with no decisions in the loop body:

```
sum = 0;
for (odd = 1; odd < n; odd+=2)
        sum += odd;
```



Case Study:

Problem: Collecting area for Solar-Heated House

- Area depends on several factors
  - the average number of heating degree days for each month
    - the product of the average difference between inside and outside temperatures and the number of days in the month
  - the average solar insolation for each month
    - rate at which solar radiation falls on one square foot of a given location
  - heating requirement per square foot of floor space
  - floor space
  - efficiency of the collection method

Case Study:

- The formula for the desired collecting area (A)
  A = heat loss / energy source
- *heat loss* is the product of the heating requirement, the floor space, and the heating degree days.
- *energy resource* is the product of the efficiency of the collection method, the average solar insolation per day and the number of days.

- Two data files
  - hdd.txt contains numbers representing the *average* heating degree days for each months.
  - solar.txt contains the average solar insolation for each month

Case Study:

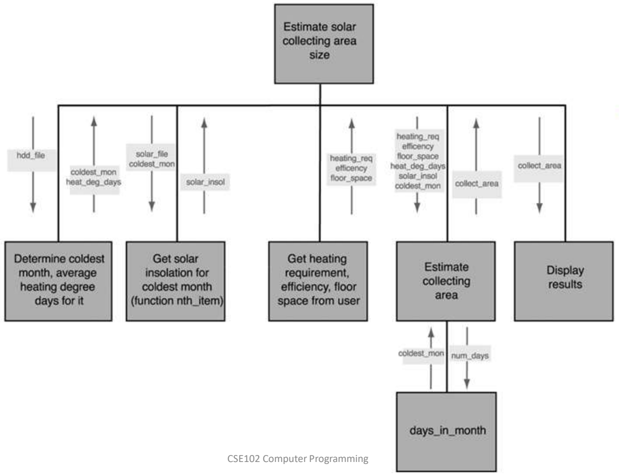
Problem Inputs
Average heating degree days file
Average solar insolation file
heat_deg_days /* average heating degree days for coldest month */ coldest_mon /* coldest month (number 1..12)
solar_insol /* average daily solar insolation for coldest month*/ heating_req
/* Btu/degree day Ft^2 */
efficiency /* % of solar insolation converted to usable heat */ floor_space /* square feet */

Program Variables
energy_resrc /* usable solar energy available in coldest month
(Btus obtained from 1 Ft^2 of collecting area) */

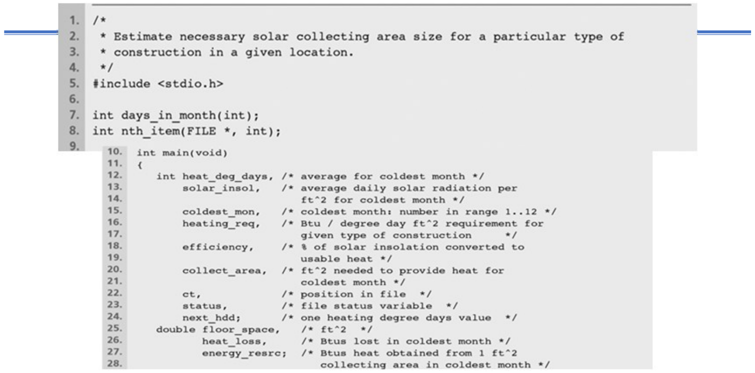
Problem Outputs
heat_loss /* Btus of heat lost by structure in coldest month */ collect_area /* approximate size Ft^2 of collecting area needed */

Case Study:

- Algorithm
  1. Determine the coldest month and the average heating degree days for this month.
  2. Find the average daily solar insolation per Ft^2 for the coldest month.
  3. Get the other problem inputs from the user :
     heating_req, efficiency, floor_space.
  1. Estimate the collecting area needed.
  2. Display results.

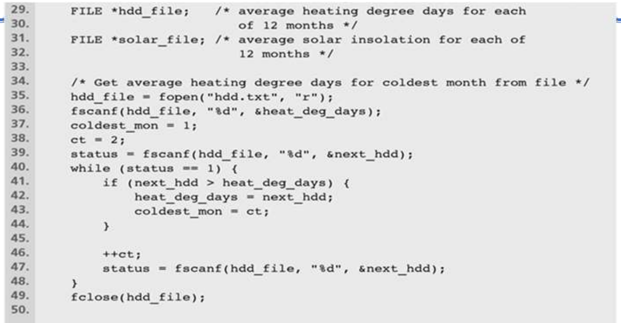

Estimate solar collecting area size
hdd_file
coldest_mon heat_deg_days
solar_file coldest_mon
solar_insol
heating_req efficency floor_space
heating_req efficency floor_space heat_deg_days solar_insol coldest_mon
collect_area
collect_area
Determine coldest month, average heating degree days for it
Get solar insolation for coldest month (function nth_item)
Get heating requirement, efficiency, floor space from user
Estimate collecting area
Display results
coldest_mon
num_days
days_in_month

Program to Approximate Solar Collecting Area Size



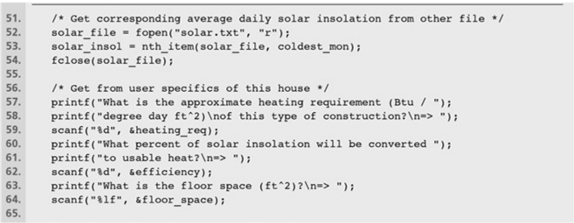
```c
/*
 * Estimate necessary solar collecting area size for a particular type of
 * construction in a given location.
 */
#include <stdio.h>

int days_in_month(int);
int nth_item(FILE *, int);

int main(void)
{
   int heat_deg_days, /* average for coldest month */
       solar_insol,   /* average daily solar radiation per
                         ft^2 for coldest month */
       coldest_mon,   /* coldest month: number in range 1..12 */
       heating_req,   /* Btu / degree day ft^2 requirement for
                         given type of construction     */
       efficiency,    /* % of solar insolation converted to
                         usable heat */
       collect_area,  /* ft^2 needed to provide heat for
                         coldest month */
       ct,            /* position in file  */
       status,        /* file status variable  */
       next_hdd;      /* one heating degree days value  */
   double floor_space,   /* ft^2  */
          heat_loss,     /* Btus lost in coldest month */
          energy_resrc;  /* Btus heat obtained from 1 ft^2
                            collecting area in coldest month */
```

65



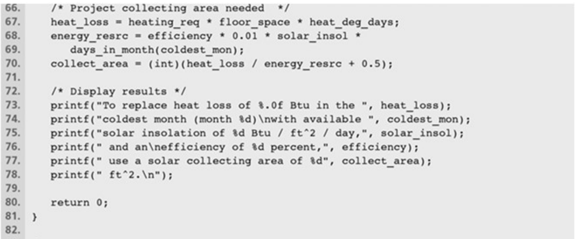
```c
    FILE *hdd_file;   /* average heating degree days for each
                         of 12 months */
    FILE *solar_file; /* average solar insolation for each of
                         12 months */

    /* Get average heating degree days for coldest month from file */
    hdd_file = fopen("hdd.txt", "r");
    fscanf(hdd_file, "%d", &heat_deg_days);
    coldest_mon = 1;
    ct = 2;
    status = fscanf(hdd_file, "%d", &next_hdd);
    while (status == 1) {
        if (next_hdd > heat_deg_days) {
            heat_deg_days = next_hdd;
            coldest_mon = ct;
        }

        ++ct;
        status = fscanf(hdd_file, "%d", &next_hdd);
    }
    fclose(hdd_file);
```

66



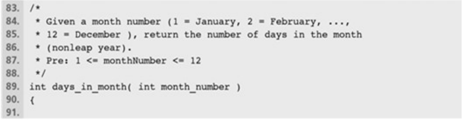
```c
    /* Get corresponding average daily solar insolation from other file */
    solar_file = fopen("solar.txt", "r");
    solar_insol = nth_item(solar_file, coldest_mon);
    fclose(solar_file);

    /* Get from user specifics of this house */
    printf("What is the approximate heating requirement (Btu / ");
    printf("degree day ft^2)\nof this type of construction?\n=> ");
    scanf("%d", &heating_req);
    printf("What percent of solar insolation will be converted ");
    printf("to usable heat?\n=> ");
    scanf("%d", &efficiency);
    printf("What is the floor space (ft^2)?\n=> ");
    scanf("%lf", &floor_space);
```

67

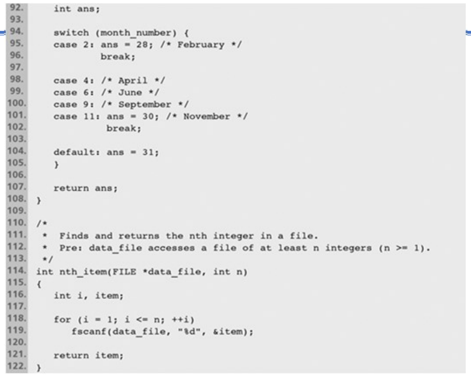


```c
    /* Project collecting area needed  */
    heat_loss = heating_req * floor_space * heat_deg_days;
    energy_resrc = efficiency * 0.01 * solar_insol *
       days_in_month(coldest_mon);
    collect_area = (int)(heat_loss / energy_resrc + 0.5);

    /* Display results */
    printf("To replace heat loss of %.0f Btu in the ", heat_loss);
    printf("coldest month (month %d)\nwith available ", coldest_mon);
    printf("solar insolation of %d Btu / ft^2 / day,", solar_insol);
    printf(" and an\nefficiency of %d percent,", efficiency);
    printf(" use a solar collecting area of %d", collect_area);
    printf(" ft^2.\n");

    return 0;
}
```

68



```
/*
 * Given a month number (1 = January, 2 = February, ...,
 * 12 = December ), return the number of days in the month
 * (nonleap year).
 * Pre: 1 <= monthNumber <= 12
 */
int days_in_month( int month_number )
{

```



```
   int ans;

   switch (month_number) {
   case 2: ans = 28; /* February */
           break;

   case 4: /* April */
   case 6: /* June */
   case 9: /* September */
   case 11: ans = 30; /* November */
            break;

   default: ans = 31;
   }

   return ans;
}

/*
 *  Finds and returns the nth integer in a file.
 *  Pre: data_file accesses a file of at least n integers (n >= 1).
 */
int nth_item(FILE *data_file, int n)
{
   int i, item;

   for (i = 1; i <= n; ++i)
      fscanf(data_file, "%d", &item);

   return item;
}
```

Altering Normal Operation of a Loop

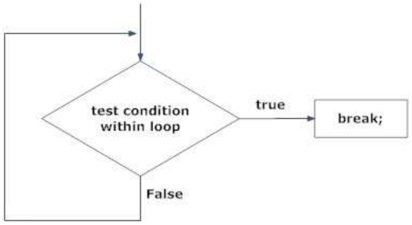
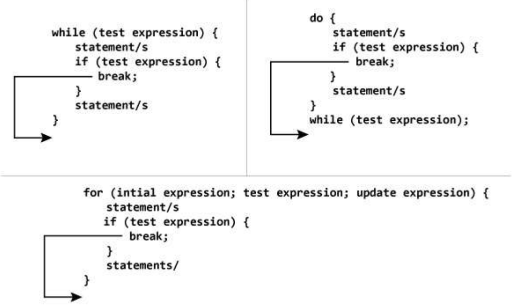


Figure: Flowchart of break statement

Altering Normal Operation of a Loop



```
while (test expression) {
   statement/s
   if (test expression) {
      break;
   }
   statement/s
}
```

```
do {
   statement/s
   if (test expression) {
      break;
   }
   statement/s
}
while (test expression);
```

```
for (intial expression; test expression; update expression) {
   statement/s
  if (test expression) {
      break;
   }
   statements/
}
```

NOTE: The break statment may also be used inside body of else statement.

Altering Normal Operation of a Loop

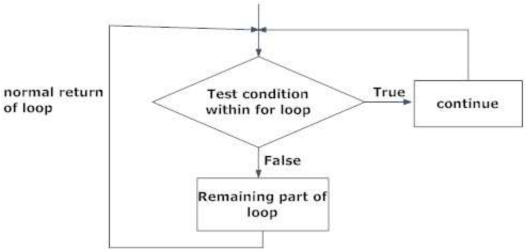
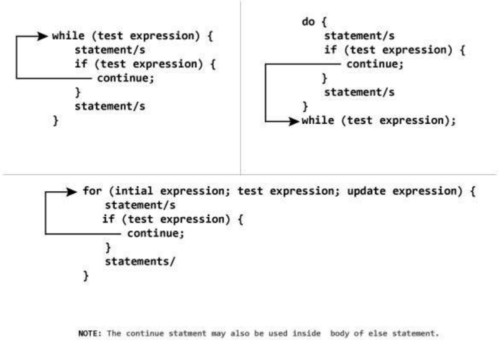


Fig: Flowchart of continue statement

Altering Normal Operation of a Loop



```
while (test expression) {
    statement/s
    if (test expression) {
        continue;
    }
    statement/s
}
```

```
do {
    statement/s
    if (test expression) {
        continue;
    }
    statement/s
}
while (test expression);
```

```
for (intial expression; test expression; update expression) {
    statement/s
   if (test expression) {
        continue;
    }
    statements/
}
```

NOTE: The continue statment may also be used inside body of else statement.

How to Debug and Test Programs

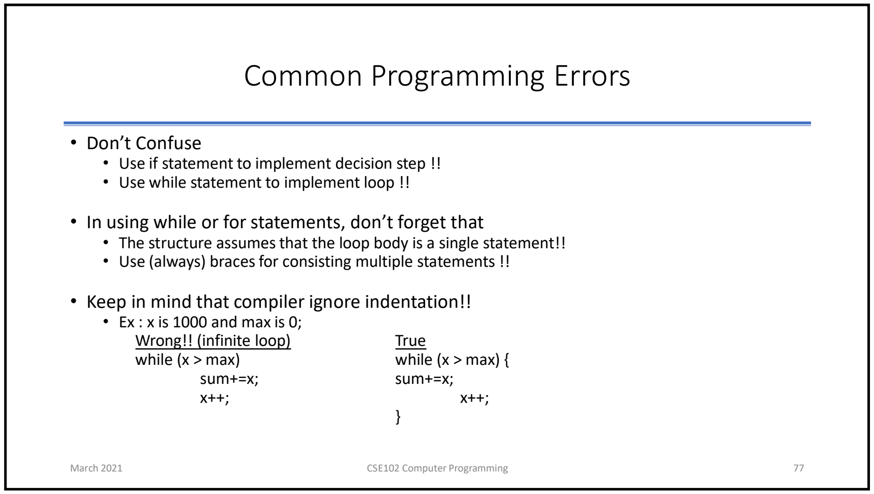
- Error Types:
  - syntax errors
  - run time errors
  - logic errors
- run-time error or logic error is usually not obvious
  - you may spend considerable time and energy locating it.
- Method:
  - examine the program output and determine program part generating incorrect results
  - focus on the statements and try to determine the fault
- OR
  - Use Debugger programs
  - Debug without debugger

Common Programming Errors

Of-by-one Loop Errors

- A common logic error with loops
  - loop executes one more time or one less time than required
- In sentinel-controlled loops, an extra repetition is more dangerous.
- In counting loops, the initial and final values of counter should be correct and the loop repetition condition should be right.
- Ex: the following loop body executes n + 1 times instead of n times.

```
for (i=0; i <= n; ++i)
        sum += i;
```

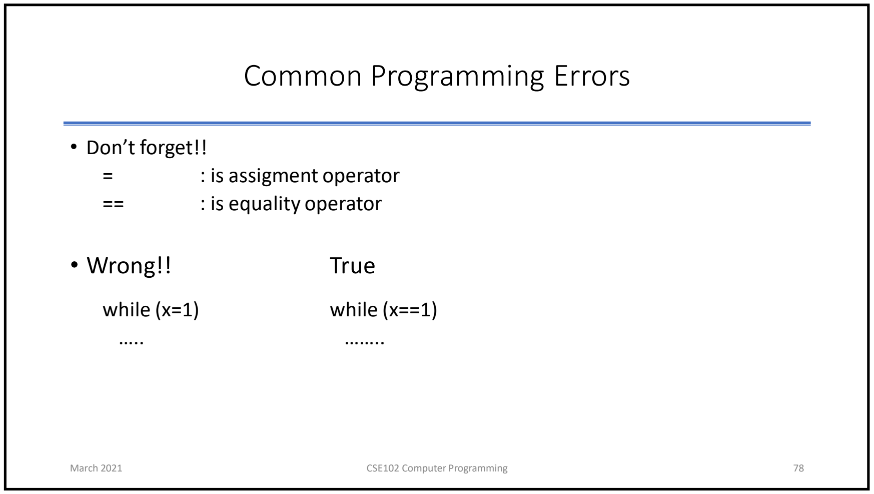


Common Programming Errors

- Don’t Confuse
  - Use if statement to implement decision step !!
  - Use while statement to implement loop !!
- In using while or for statements, don’t forget that
  - The structure assumes that the loop body is a single statement!!
  - Use (always) braces for consisting multiple statements !!
- Keep in mind that compiler ignore indentation!!
  - Ex : x is 1000 and max is 0;

```
Wrong!! (infinite loop)          True
while (x > max)                  while (x > max) {
        sum+=x;                  sum+=x;
        x++;                             x++;
                                 }
```

77



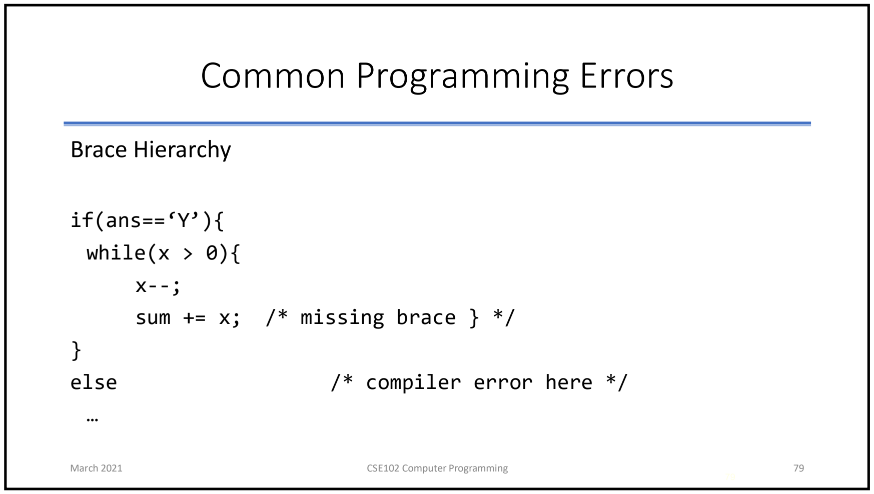
Common Programming Errors

- Don’t forget!!

```
=        : is assigment operator
==       : is equality operator
```

```
• Wrong!!              True
  while (x=1)          while (x==1)
    .....                ........
```

78

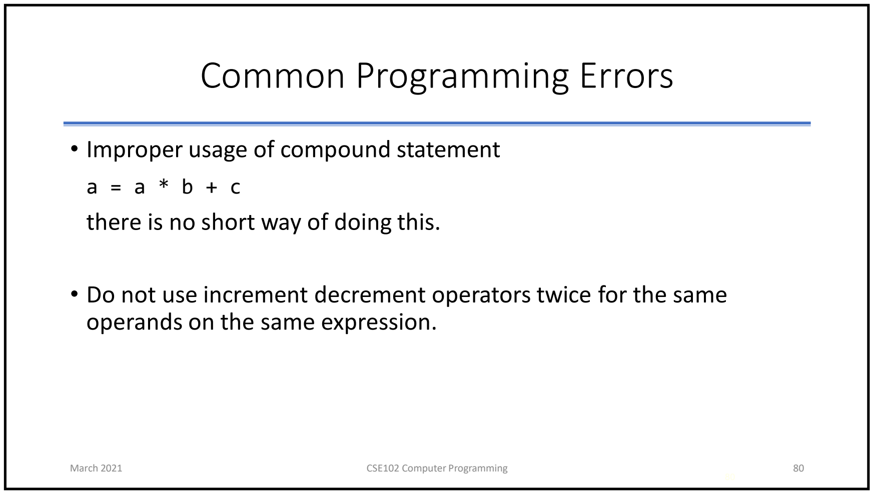


Common Programming Errors

Brace Hierarchy

```
if(ans==‘Y’){
 while(x > 0){
     x--;
     sum += x;  /* missing brace } */
}
else                 /* compiler error here */
 …
```

79



Common Programming Errors

- Improper usage of compound statement

  `a = a * b + c`

  there is no short way of doing this.

- Do not use increment decrement operators twice for the same operands on the same expression.

80

Thanks for listening!

81