

# Programming in C



## Looping Subtasks



# Looping Subtasks

- We will examine some basic algorithms that use the while and if constructs. These subtasks include
  - Reading unknown quantity of data
  - Counting things
  - Accumulating (summing) totals
  - Searching for specific values
  - Finding extreme values

# Looping Subtasks

- Examples will be based upon common models:

Priming Read

or

Input Count

*Initialize program state*  
*Read the first value (priming read)*  
*While (data exists)*  
    *update program state as needed*  
    *read next value(s)*  
*Output final state*

*Initialize program state*  
*While (input count OK)*  
    *update program state as needed*  
*Output final state*

- The type of state that must be maintained by the program depends on the nature of the problem and can include:
  - *indicator (true/false) variables*
  - *counter variables*
  - *sum variables*
  - *previous input value variables*

# Counter-Controlled Repetition

- Number of items is known before loop

```
// Read and print 5 test scores
int count, score;
for (count = 1; count <= 5; count++) {
    scanf("%d", &score);
    printf("Score %d is %d\n", count, score);
}
```

- Suppose the problem becomes:

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



# Sentinel-Controlled Repetition

- One way to handle an arbitrary number of input values is to have the user enter a special value to indicate the end of input.
- Such a value is a sentinel value.
  - Indicates end of valid input
  - Loop ends when sentinel value is read
  - Must choose a sentinel value that cannot be confused with a regular input value.

25  
43  
67  
96  
12  
58  
44  
-1

sentinel



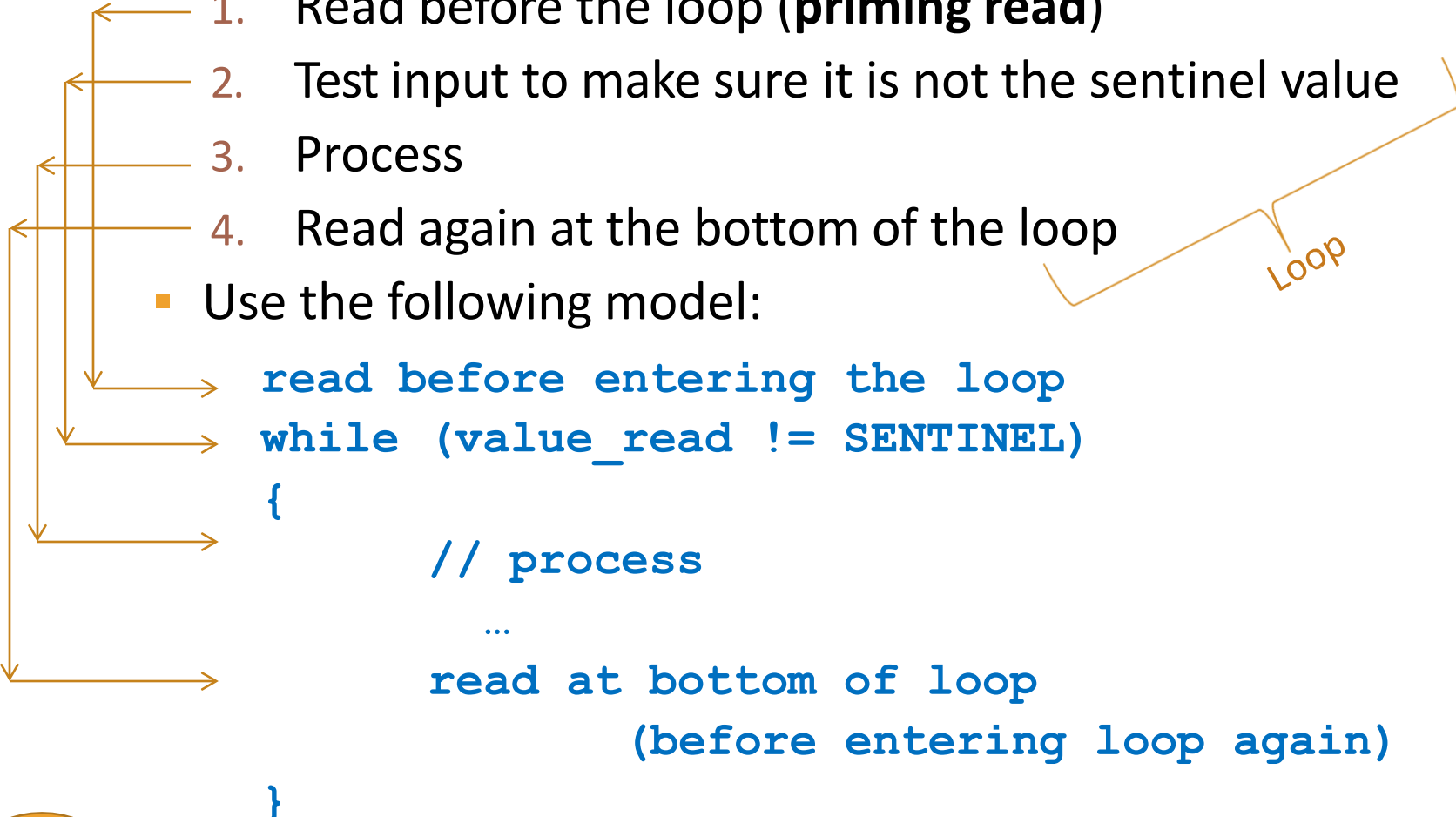
# Sentinel-Controlled Priming Read

- For sentinel-controlled loops

1. Read before the loop (**priming read**)
2. Test input to make sure it is not the sentinel value
3. Process
4. Read again at the bottom of the loop

- Use the following model:

```
read before entering the loop
while (value_read != SENTINEL)
{
    // process
    ...
    read at bottom of loop
    (before entering loop again)
}
```



# Sentinel-Controlled Loop using Priming Read

25  
43  
67  
96  
12  
58  
44  
-1

**sentinel**



```
// Read and print numbers using priming read
int num;

scanf("%d", &num);           // Priming read
while (num != -1) {         // Sentinel is -1
    printf("%d\n", num);
    scanf("%d", &num);       // Read another number
}
```

# Sentinel-Controlled Loop using Input Count

25  
43  
67  
96  
12  
58  
44  
-1

sentinel



```
// Read and print numbers using input count
int inputCount;           // Items read
int num;

while (scanf("%d", &num) == 1 && num != -1) {
    // Sentinel is -1
    printf("%d\n", num);
}
```



# Example of sentinel-controlled loop

25 43  
67 96  
12 58  
44 99  
-1

**sentinel**



```
// Read pairs and print sums
int num1, num2, sum;

scanf("%d", &num1);           // Priming read
while (num1 != -1) {         // Sentinel is -1
    scanf("%d", &num2);       // Read second number
    sum = num1 + num2;
    printf("%d + %d = %d\n", num1, num2, sum);
    scanf("%d", &num1);       // Read another first number
}
```

# Processing an arbitrary number of pairs

- Sometimes it is not possible to find a sentinel value
- We can use
  - End-of-input controlled loops
    - Uses return from scanf
    - Can be fooled by invalid data
  - End-of-file controlled loops
    - Uses function feof



# End of Data

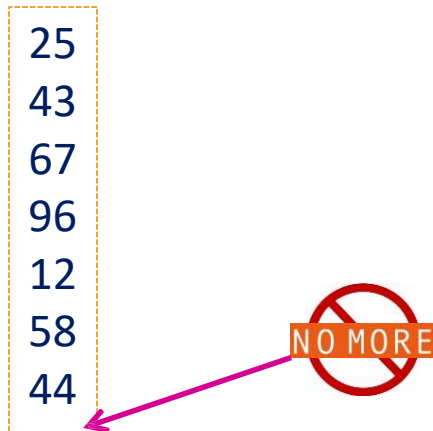
- Hardware & Software
  - End-Of-File
    - Keyboard
      - Ctrl-d (Does not work on Mac!)

```
25 43  
67 96  
12 58  
44 99  
Ctrl-d
```



The End Is Here!

# Example: End-of-input controlled loop using items read & priming read



```
// Read and print using items read & priming read
int inputCount;           // Items read
int num;

inputCount = scanf("%d", &num); // Priming read
while (inputCount == 1) {     // Check count
    printf("%d\n", num);
    inputCount = scanf("%d", &num); // Read another number
}
```

# Example: End-of-input controlled loop using just items read

```
// Read and print using just items read
int num;

while (scanf("%d", &num) == 1) { // Check count
    printf("%d\n", num);
}
```

25  
43  
67  
96  
12  
58  
44

```
//or
while (scanf("%d", &num) != EOF) { // Check for EOF
    printf("%d\n", num);
}
```



# Example: End-of-input controlled loop using number of items read

25 43  
67 96  
12 58  
44 99



```
// Read pairs and print sums using items read
int num1, num2, sum;

while (scanf("%d %d", &num1, &num2) == 2) { // Check items read
    sum = num1 + num2;
    printf("%d + %d = %d\n", num1, num2, sum);
}
```

# Detecting End-of-File

- Function: `feof`
  - Syntax: `feof(file-pointer)`
    - Returns true or false
    - Standard input: `feof(stdin)`
  - Use in a while loop -  
`while (!feof(stdin))`

# Example: End-of-file controlled loop

25  
43  
67  
96  
12  
58  
44

End of File



```
// Read and print numbers using EOF
int num;

scanf("%d", &num);           // Priming read
while (!feof(stdin)) {      // Check for EOF
    printf("%d\n", num);
    scanf("%d", &num);       // Read another number
}
```



# Example: end-of-file controlled loop

25 43  
67 96  
12 58  
44 99

End of File

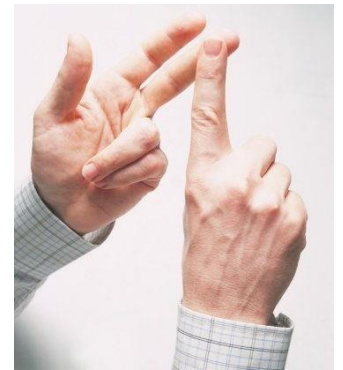


```
// Read pairs and print sums using end-of-file
int num1, num2, sum;

scanf("%d", &num1);           // Priming read
while (!feof(stdin)) {       // Check for EOF
    scanf("%d", &num2);       // Read second number
    sum = num1 + num2;
    printf("%d + %d = %d\n", num1, num2, sum);
    scanf("%d", &num1);       // Read another first number
}
```

# Looping Subtask: Counting

- Example: Find the number of scores in a file
  - Here the program state that must be maintained is a *counter* that maintains the number of scores that have been read so far.
- Steps
  - Declare an int variable for the count
  - Initialize the count to zero
  - Increment the count in the body of the loop



# Looping Subtask: Counting

```
// Print score count w/priming read
int scoreCount;           // counter
int score;

scoreCount = 0;           // initialize counter

printf("Enter first score or ctrl-d to end: ");
scanf("%d", &score);
while (!feof(stdin)) {
    scoreCount++;         // increment counter
    scanf("%d", &score);
    printf("Enter next score or ctrl-d to end: ");
}

printf("Score count is %d\n", scoreCount);
```

# Looping Subtask: Counting

```
// Print score count w/scanf in while
int scoreCount;           // counter
int score;

scoreCount = 0;           // initialize counter

printf("Enter first score or ctrl-d to end: ");
while (scanf("%d", &score) == 1) {
    scoreCount++;         // increment counter
    printf("Enter next score or ctrl-d to end: ");
}

printf("Score count is %d\n", scoreCount);
```

# Looping Subtask: Counting

```
// Print score count w/for
int scoreCount;           // counter
int score;

scoreCount = 0;          // initialize counter

printf("Enter first score or ctrl-d to end: ");
for (scoreCount = 0; scanf("%d", &score) == 1, scoreCount++)
    printf("Enter next score or ctrl-d to end: ");

printf("Score count is %d\n", scoreCount);
```

# Looping Subtask: Counting

```
// Print score count w/for & no prompts
int scoreCount;           // counter
int score;

scoreCount = 0;           // initialize counter

for (scoreCount = 0; scanf("%d", &score) == 1, scoreCount++)
    /*null*/ ;

printf("Score count is %d\n", scoreCount);
```

# Counting Example

- What if we want to print the number of passing scores (scores  $\geq 70$ )?
  - We need a mechanism that allows us to count only if the score is greater than or equal to 70
  - Use *if* stmt

# Looping Subtask: Counting

```
// Print passing score count
int passCount;           // passing counter
int score;

passCount = 0;          // initialize counter

scanf("%d", &score);
while (!feof(stdin)) {
    if (score >= 70)
        passCount++;    // increment pass counter
    scanf("%d", &score);
}

printf("Passing score count is %d\n", passCount);
```



# Counting Example

- What if we want to print the number of passing scores (scores  $\geq 70$ ) and the number of failing scores?
  - Use *if-else*

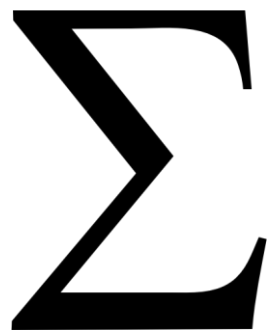
# Looping Subtask: Counting

```
// Print passing and failing score count
int passCount;           // passing counter
int failCount:          // failing counter
int score;

passCount = 0;           // initialize counters
failCount = 0;

scanf("%d", &score);
while (!feof(stdin)) {
    if (score >= 70)
        passCount++;     // increment pass counter
    else
        failCount++;     // increment fail counter
    scanf("%d", &score);
}

printf("Passing score count is %d\n", passCount);
printf("Failing score count is %d\n", failCount);
```



## Looping Subtask: Accumulation (Summing)

- The state that must be maintained is the sum of all values that have been seen so far.
  - Declare a variable to hold the sum (accumulator)
  - Initialize the sum to zero
  - In the body of the loop, add the new value to the sum

# Accumulating Example

```
// Print score sum
int scoreSum;           // total accumulator
int score;

scoreSum = 0;          // initialize total

scanf("%d", &score);
while (!feof(stdin)) {
    scoreSum += score; // add score to total
    scanf("%d", &score);
}

printf("Score total is %d\n", scoreSum);
```

# Counting & Accumulating Example

- Problem
  - *A class of ten students took a quiz.*
  - *The grades (integers in the range 0 to 100) for this quiz are available to you.*
  - *Determine the class average on the quiz.*
- Hint: Requirements for an average
  - Count of number of items
  - Sum of the items



# Counting & Accumulating Example

- Pseudocode:

*Set total to zero*

*Set grade counter to one*

*While grade counter is less than or equal to ten*

*Input the next grade*

*Add the grade into the total*

*Add one to the grade counter*

*Set the class average to the total divided by ten*

*Print the class average*

EXCELLENT



# Looping Subtasks: Searching



- Need a variable to indicate whether or not the program has encountered the target value, call it *found*
- Initialize *found* to 0 (false)
- Each time through the loop, check to see if the current value equals the target value
  - If so, assign 1 to *found*

# Searching Exercise

Write a C program that

1. Reads a target score at the beginning of the file
2. Reads a set of scores and determines if the target score is in the set of scores

3. If found prints

**Target ## was found**

otherwise prints

**Target ## was not found**



# Looping Subtasks: Searching

```
// Determine if target score is found
int score, target;
int found = 0;           // found = false

scanf("%d", &target);

scanf("%d", &score);
while (!feof(stdin)) {
    if (score == target)
        found = 1;       // found = true
    scanf("%d", &score);
}

if (found)
    printf("Target %d was found\n", target);
else
    printf("Target %d was not found\n", target);
```

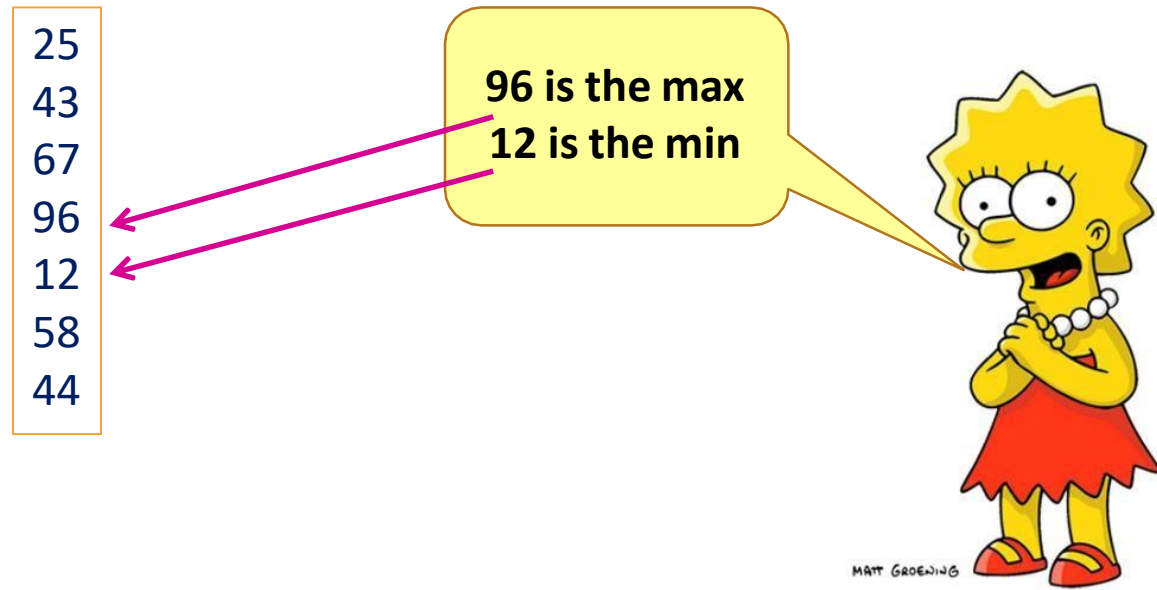
# Searching Improvement

- Stop searching if target has been found

```
scanf("%d", &score);
while (!feof(stdin) && !found) {
    // stop if EOF or target found
    if (score == target)
        found = 1;           // found = true
    scanf("%d", &score);
}
```

# Looping Subtasks: Finding Extremes

- Finding Extreme Values (e.g. maximum, minimum)
  - Need a variable (such as maxValue) to remember the most extreme value encountered so far



# Looping Subtasks: Finding Extremes

- Finding Extreme Values (e.g. maximum, minimum)
  - Initialize the max**Value** (min**Value**) to some value
    - max**Value**: Lower value than any data
    - min**Value**: Higher value than any data
    - Or for both: The first data value
  - For each data item
    - Compare the current value to max**Value** (or min**Value**)
    - If the current value is  $>$  max**Value** ( $<$  min**Value**), replace max**Value** (min**Value**) with the current value.

# Extremes Exercise

Write a C program that

1. Reads a set of scores from a file
2. Determines and prints the maximum score

# Looping Subtasks: Finding Extremes

```
// Determine maximum score
int score, maxScore;

scanf("%d", &score);
maxScore = score;           // initialize max

while (!feof(stdin)) {
    if (score > maxScore)
        maxScore = score;   // reset max
    scanf("%d", &score);
}

printf("Maximum score is %d\n", maxScore);
```

# Programming in C



## Looping Subtasks



***T H E E N D***