Problem Solving & Program Design in C

Chapter 5:
Repetition and Loop Statements

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
- Use relational and logical expressions to control loops until condition is met

Comparison of Loop Types

- counting loop
  - we can determine before loop execution exactly how many loop repetitions will be needed to solve the problem
    - while, for
- sentinel-controlled loop
  - input of a list of data of any length ended by a special value
    - while, for, do-while

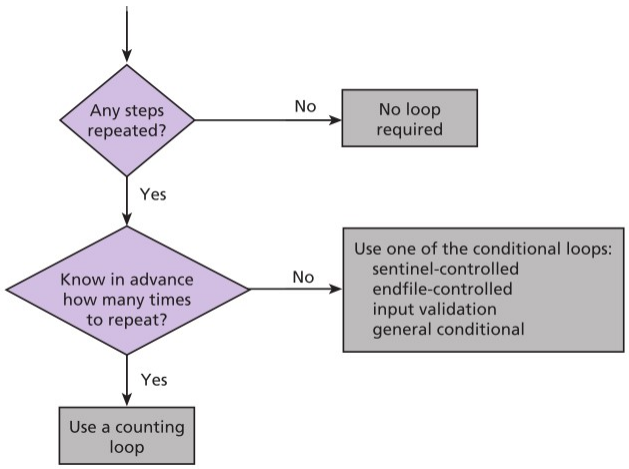
Comparison of Loop Types

- endfile-controlled loop
  - input of a single list of data of any length from a data file
    - while, for, do-while
- input validation loop
  - repeated interactive input of a data value until a value within the valid range is entered
    - do-while
- general conditional loop
  - repeated processing of data until a desired condition is met
    - while, for

Choosing a Loop Type

FIGURE 5.1

Flow Diagram of Loop Choice Process



Counting Loops

- counter-controlled loop
  - a.k.a. counting loop
  - a loop whose required number of iterations can be determined before loop execution begins
- loop repetition condition
  - the condition that controls loop repetition
- loop control variable
  - the variable whose value controls loop repetition
- infinite loop
  - a loop that executes forever

The While Loop for Counting

while (loop repetition condition)
    statement;

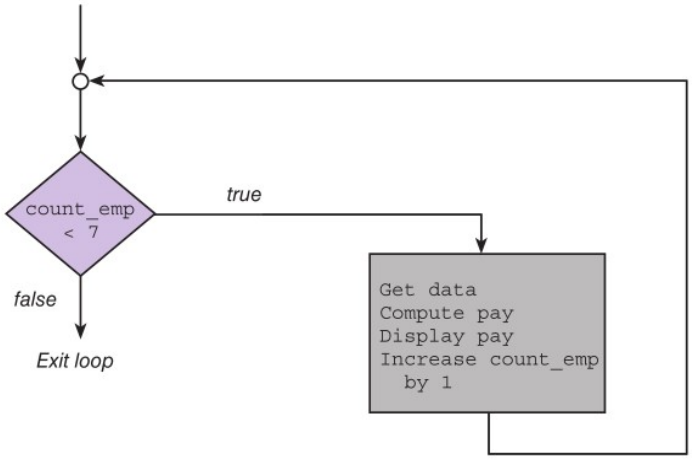
- Using a loop
  - Initialize control variable
  - Condition to terminate loop
  - Change control variable so loop eventually terminates
- Loop invariant
  - Ensure that all conditions correct before loop starts
  - Ensure that everything in loop body correct
  - Ensure that loop terminates

```
int i = 0;

while(i < 5)
   i++;

printf("i = %d\n", i);
```

Calculate Pay for 7 Employees

```
int count_emp = 0, hours = 0;
double rate = 0.0, pay = 0.0;

while(count_emp < 7){
   printf("Hours> ");
   scanf("%d", &hours);
   printf("Rate> ");
   scanf("%lf", &rate);
   pay = hours * rate;
   printf("Pay is $%6.2f\n", pay);
   count_emp += 1;
}
```

Flowchart for Pay 7 Employees



The While Loop for Accumulating

- Accumulator
  - a variable used to store a value being computed in increments during the execution of a loop
- Sum or product computed in a loop
- Make sure to initialize all variables
- Note {…} for compound statement loop body

```
int i = 0;
double sum = 0.0;
double input = 0.0;

while(i < 5){
   printf("Enter value: ");
   scanf("%lf", &input);
   sum += input;
   i++;
}

printf("sum = %.2f\n", sum);
```

Accumulate Total Pay for Employees

```
int count_emp = 0, hours = 0;
double rate = 0.0, pay = 0.0, total = 0.0;

while(count_emp < 7){
   printf("Hours> ");
   scanf("%d", &hours);
   printf("Rate> ");
   scanf("%lf", &rate);
   pay = hours * rate;
   printf("Pay is $%6.2f\n", pay);
   total += pay;
   count_emp += 1;
}
printf("Total is $%6.2f\n", total);
```

General Conditional Loop

1. Initialize loop control variable.
2. As long as exit condition hasn't been met
   3. Continue processing

```
int product = 1, item  = 0;
  while(product < 10000){
     printf("%d\n", product);
     printf("Enter next item> ");
     scanf("%d", &item);
     product *= item;
  }
```

Loop Control Components

- initialization of the loop control variable
- test of the loop repetition condition
- change (update) of the loop control variable

- the for loop supplies a designated place for each of these three components

The For Loop

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

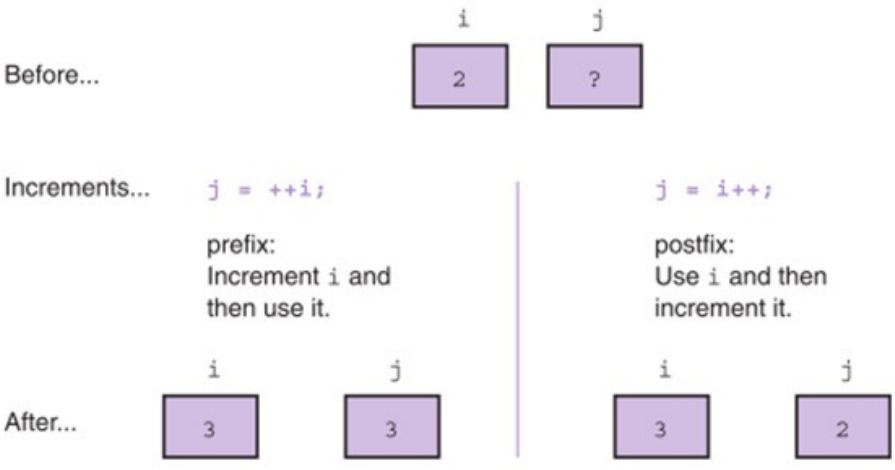
- For loop contains
  - Initialization: i = 0
  - Test to stop: i < n
  - Change control variable: i++
  - Note ; in () not ,

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

Increment and Decrement Operators

- counter = counter + 1;
  count += 1;
  counter++;
- counter = counter – 1;
  count -= 1;
  counter- -;
- side effect
  - a change in the value of a variable as a result of carrying out an operation
  - Prefix (++count) increment before use
  - postfix (count++) use before increment

Increment and Decrement Operators



Computing Factorial

- loop body executes for decreasing value of i from n through 2
- each value of i is incorporated in the accumulating product
- loop exit occurs when i is 1

```
int factorial(int n){
   int i = n, product = 1;
   if(n == 0)
      return product;
   for(i=n; i>1; --i)
      product *= i;
   return product;
}
```

Conversion of Celsius to Fahrenheit

- example shows conversions from 10 (CBEGIN) degree Celsius to -5 (CLIMIT) degrees Celsius
- loop update step subtracts 5 (CSTEP) from Celsius
  - accomplished by decreasing the value of the counter after each repetition
- loop exit occurs when Celsius becomes less than CLIMIT

Conversion of Celsius to Fahrenheit

```
#define CBEGIN 10
#define CLIMIT -5
#define CSTEP 5

void convert(){
   int celsius = 0;
   double fahrenheit = 0.0;
   for(celsius = CBEGIN;
      celsius >= CLIMIT;
      celsius -= CSTEP){
         fahrenheit = 1.8 * celsius + 32.0;
         printf("%6c%3d%8c%7.2f\n", ' ', celsius, ' ', fahrenheit);
      }
   return;
}
```

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

```
#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_BARREL 42  // Number of gallons per barrel
// Prototype
double monitor_gas(double min_supply, double start_supply);

int main(){
   // Input – initial supply in barrels
   double start_supply;
  // Min barrels left without warning
   double min_supply = MIN_PCT / 100.0 * CAPACITY;
   // Get initial supply
   printf("Number of barrels currently in tank> ");
   scanf("%lf", &start_supply);
   // Subtract amount removed and display remaining until min supply reached
   double current = monitor_gas(min_supply, start_supply);
   // Min supply reached – print 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;
}
```

```
// Computes and displays amount of gas remaining after each delivery
// Pre-condition: min_supply and start_supply defined
// Post-condition: Returns supply available after min supply reached
double monitor_gas(double min_supply, double start_supply){
   // Input (current delivery), input in gallons and output (current supply)
   double remove_gals, remove_barrels, current;
   // Loop until min supply reached
   for(current = start_supply;
      current >= min_supply;
      current -= remove_barrels){
         // Print barrels available
         printf("%.2f barrels are available.\n\n", current);
         // Get current delivery in gallons
         printf("Enter number of gallons to be removed> ");
         scanf("%lf", &remove_gals);
         // Convert current delivery to barrels
         remove_barrels = remove_gals / GALS_PER_BARREL;
         // Print gallons and barrels removed
         printf("After removal of %.2f gallons (%.2f barrels),\n",
               remove_gals, remove_barrels);
      }
   // Return current supply in barrels
   return current;
}
```

Loop Design

- Sentinel-Controlled Loops
  - Sentinel value: an end marker that follows the last item in a list of data ☺
- Endfile-Controlled Loops
  - Stop loop at end of file ☺
- Infinite Loops on Faulty Data
  - Bad ☹

Sentinel Loop Design

- Correct Sentinel Loop
  1. Initialize sum to zero.
  2. Get first score.
  3. while score is not the sentinel
     4. Add score to sum.
     5. Get next score
- Incorrect Sentinel Loop
  1. Initialize sum to zero.
  2. while score is not the sentinel
     3. Get score
     4. Add score to sum.

Sentinel Loop Example

```
#include <stdio.h>

#define SENTINEL -99 // Value that marks the end of input

int main(){
   // Output (sum of scores) and input (current score)
   int sum = 0, score = 0;
   // 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(or %d to quit)> ", SENTINEL);
      scanf("%d", &score);  // Get next score
   }
   printf("Sum 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*

Endfile-Controlled Loop Example

```
#include <stdio.h>

int main(){
   // Sum of scores input, current score and scanf status
   int sum = 0, score = 0, input_ststus = 0;
   printf("Scores\n");
   input_ststus = scanf("%d", &score); // Read first score
   while(input_ststus != EOF){
      printf("%5d\n", score); // Print current score
      sum += score;  // Accumulate sum of scores
      input_ststus = scanf("%d", &score); // Read next score
   }
   // Print sum of scores
   printf("Sum of exam scores is %d\n", sum);
   return 0;
}
```

Loops and Selection Structures

- Loops can contain selection structures
  - They can be nested
- If-else
- Else-if chains
- Switch
- Keywords break and continue
  - break – break out of a loop
  - continue – go to beginning of loop body

Loop Using If-Else and Break

```
#include <stdio.h>

#define SENTINEL -99 // Value that marks the end of input

int main(){
   // Output (sum of scores) and input (current score)
   int sum = 0, score = 0;
   while(1){  // Loop forever
      // Get a score
      printf("Enter a score(or %d to quit)> ", SENTINEL);
      scanf("%d", &score);
      // Check for SENTINEL
      if(score == SENTINEL)
         break; // Break out of loop
      else
         sum += score; // Accumulate sum
   }
   printf("Sum of exam scores is %d\n", sum);
   return 0;
}
```

Loop Using If-Else and Continue

```
#include <stdio.h>

int main(){
   // Iterator and number
   int i = 0, n = 0;
   // Get n from user
   printf("Enter n> ");
   scanf("%d", &n);
   // Loop and print even numbers from 0 to n
   for(i=0; i<=n; i++){
      // If odd n, restart loop
      if(i%2 == 1)
         continue;
      // Print n
      printf("i = %d\n", i);
   }
   return 0;
}
```

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

Print from a Nested Loop

```
#include <stdio.h>

int main(){
   // Loop iterators
   int i = 0, j = 0;
   // Print labels
   printf("         i  j\n");
   for(i=0; i<4; i++){ // Loop through i
      printf("Outer %6d\n", i); // Print outer loop iterator
      for(j=0; j<i; j++) // Loop through j
         printf("Inner %9d\n", j); // Print inner loop iterator
   }
   return 0;
}
```

do-while Statement

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

- 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.

```
int status = 0, num = 0;
do {
   status = scanf("%d", &num);
} while(status > 0 && (num%2) != 0);
printf("status = %d num = %d\n",
status, num);
```

Get an Input Integer from 1 to 5

```
#include <stdio.h>
#define LOW 1
#define HIGH 5

int main(){
   int n = 0;
   // Get a number from user between LOW and HIGH, inclusive
   do{
      // Get number
      printf("Enter a number from 1 to 5> ");
      scanf("%d", &n);
      // Print message for bad entry
      if(n < LOW || n > HIGH)
         printf("%d is not a valid choice\n", n);
   }while(n < LOW || n > HIGH); // While bad entry
   // Print correctly entered number
   printf("You entered: %d\n", n);
   return 0;
}
```

Flag-Controlled Loops for Input Validation

- Sometimes a loop repetition condition becomes so complex that placing the full expression in its usual spot is awkward
- Simplify the condition by using a **flag**

- flag
  - a type int variable used to represent whether or not a certain event has occurred
  - 1 (true) and 0 (false)

Nested Do-While Loop with Error Flag

```
#include <stdio.h>
#define MIN 1
#define MAX 5

int main(){
   // Input value, scanf status, error flag and skip bad chars
   int in_val = 0, status = 0, error = 0;
   char skip_char;
   // Get a number from user between LOW and HIGH, inclusive
   do{
      error = 0; // Reset error flag to no error
      // Get number from user
      printf("Enter a number from %d to %d> ", MIN, MAX);
      status = scanf("%d", &in_val);
      // Print scanf error for non-int entry
      if(status != 1){
         error = 1; // Set error flag to error
         scanf("%c", &skip_char);
         printf("Invalid input %c\n", skip_char);
      }
      // Print message for int entry out of range
      else if(in_val < MIN || in_val > MAX){
         error = 1; // Set error flag to error
         printf("%d is not a valid choice\n", in_val);
      }
      // Remove all invalid chars from input stream
      do{
         scanf("%c", &skip_char);
      }while(skip_char != '\n');
   }while(error); // While error flag is set to bad entry
   // Print correctly entered number
   printf("You entered: %d\n", in_val);
   return 0;
}
```

Infinite Loop Errors

- Loop runs forever when unintended
- Usually because a condition does not change in loop control variable

```
#include <stdio.h>

int main(){
   int i = 0, n = 1;
   while(i<10){
      printf("n = %d\n", n);
      n = n * 2;
   }
   return 0;
}
```

Off-by-One Loop Errors

- A fairly common logic error in programs with loops is a loop that executes on more time or one less time than required.
- If a sentinel-controlled loop performs an extra repetition, it may erroneously process the sentinel value along with the regular data.
- loop boundaries
  - initial and final values of the loop control variable