Repetition and Loop statement

There is repetition of steps.

We go for loops.

Know in advance how many times to repeat

Use a counting loop

Donot Know in advance how many times to repeat

sentinel-controlled

endfile-controlled

input validation

general conditional

Loop Kinds

Kind	When Used	C Implementation Structures
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
Endfile-controlled loop	*Input of a single list of data of any length from a data file*	while, for
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

Endless Loop

- The control conditions must be well defined and specified otherwise the loop will execute an infinite number of times. The loop that does not stop executing and processes the statements number of times is called as an infinite loop. An infinite loop is also called as an "Endless loop."

1. No termination condition is specified.
2. The specified conditions never meet.

Counting Loops and the while Statement

Counter-controlled loop

A loop whose required number of iterations can be determined before loop execution begins.

```
Set loop control variable to an initial value of 0 .
while loop control variable < final value
. . .
Increase loop control variable by 1 .
```

Syntax of the while Statement

(1) initialized,

(2) tested, and

(3) updated for the loop to execute properly.

loop control variable: the variable whose value controls loop repetition

loop repetition condition: the condition that controls loop repetition.

The loop is repeated when the condition is true.

The loop is exited when this condition is false.

Example

```
#include<stdio.h>
int main()
{
int num=1;                  //initializing the variable
while(num<=10)              //while loop with condition
{
printf("%d\n",num);
num++;                      //incrementing operation
}
return 0;
}
```

Example

1. A program that computes and displays the gross pay for seven employees.
Input: hours worked, rate, pay
Pay= hours * rate

2. Modify problem 1 to print company's total payroll.

3. Write a program fragment that produces this output:

```
0    1
1    2
2    4
3    8
4   16
5   32
6   64
```

4. Program to print ODD numbers from 1 to N using while loop.
5. C program to print all uppercase alphabets using while loop.
6. Multiplying a list of numbers.
7. Design a program for a calculator.

cont ...

8. Write a menu driven program that has the following outputs:

1. Factorial of a number
2. Prime or not
3. Odd or even
4. Exit

Output:

1. *Factorial*
2. *Prime or not*
3. *Odd or even*
4. *Exit*

Enter an option:

do...while loop

- A do...while loop in C is similar to the while loop except that the condition is always executed after the body of a loop. It is also called an exit-controlled loop.
- Syntax of do...while loop in C programming language:

- do {
- statements
- } while (expression);

- In a while loop, the body is executed if and only if the condition is true. In some cases, we have to execute a body of the loop at least once even if the condition is false. This type of operation can be achieved by using a do-while loop.

- In the do-while loop, the body of a loop is always executed at least once. After the body is executed, then it checks the condition. If the condition is true, then it will again execute the body of a loop otherwise control is transferred out of the loop.

- The critical difference between the while and do-while loop is that in while loop the while is written at the beginning. In do-while loop, the while condition is written at the end and terminates with a semi-colon (;)

Example: To print a table of number 2

```
#include<stdio.h>
int main()
{
int num=1;              //initializing the variable
do                      //do-while loop
{
    printf("%d\n",2*num);
    num++;
}while(num<=10);
return 0;
}
```

do...while() loop cont ...

1. Print integers between 0 and 10 using a do...while() loop.
2. Finding the average of first N natural numbers using a do...while() loop in C.
3. Print a **multiplication** table of an input number N

Compound assignment operators

Assignment statement	Compound
Variable = variable op expression	variable op = expression;
op: arithmetic operator	

Simple assignment operator	compound
emp = emp+1;	emp + = 1;
time =time – 1;	time -= 1;
prod = prod * item;	prod * = item;

Op = assignment operators +=, -=, /=, *= and %=

Problem solving

Exercise Section 5.3

For statement

SYNTAX:

```
for (initialization expression;
    loop repetition condition;
    update expression)
Statement;
```

Example

```
/* Display N asterisks. */

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

- A for loop is a more efficient loop structure in 'C' programming.

- The general structure of for loop syntax in C is as follows:
- for (initial value; condition; incrementation or decrementation)

```
 {
   statements;
}
```

- The initial value of the for loop is performed only once.
- The condition is a Boolean expression that tests and compares the counter to a fixed value after each iteration, stopping the for loop when false is returned.
- The incrementation/decrementation increases (or decreases) the counter by a set value.

For Loop Example

```
#include<stdio.h>
int main()
{
int number;
for(number=1;number<=10;number++) //for loop to print 1-10 numbers
    {
    printf("%d\n",number);       //to print the number
    }
return 0;
}
```

Increment and Decrement Operators

The value of the expression in which the **++** operator is used

depends on the **position** of the operator.

When the ++ is placed immediately in front of its operand (prefix increment),

the value of the expression is the variable's value after incrementing.

When the ++ comes immediately after the operand (postfix increment),

the expression's value is the value of the variable before it is incremented.

initial value of n: 4

printf("%3d", --n); o/p: 3

printf("%3d", n); o/p: 3

printf("%3d", n--); o/p: 4

Printf("%3d", n); o/p: 3

Programs executing For Loop

1. To find factorial of a number using For Loop.
2. Conversion of Celsius to Fahrenheit (decrement at each step by 5, Beg :10, End: -5).

   output:

Celsius	Fahrenheit
10	50.00
5	41.00
0	32.00
-5	23.00

3. Write a program to display a centimeters-to-inches conversion table. The smallest and largest number of centimeters in the table are input values. Your table should give conversions in 10-centimeter intervals. One centimeter equals 0.3937 inch.

4. Rewrite the payroll program (Fig. 5.5), moving the loop processing into a function subprogram. Return the total payroll amount as the function result.

Nested For Loop

```
for(initialization; condition; increment){
      for(initialization; condition; increment){
    //statement of inner loop;
 }
   //statement of outer loop;
 }
```

Nested for Loop cont…

1. Write a program that displays the multiplication table for numbers 0 to 9.
2. Write nests of loops that cause the following output to be displayed:

0

0 1

0 1 2

0 1 2 3

0 1 2 3 4

0 1 2 3 4 5

0 1 2 3 4

0 1 2 3

0 1 2

0 1

0

Sentinel-Controlled Loops

- The number of data items the loop should process when it begins execution is not known.
- A way should be found to signal the program to stop reading and processing new data.
- One way to do this is to instruct the user to enter a unique data value, called a sentinel value , after the last data item.
- **Sentinel value**: an end marker that follows the last item in a list of data
- The loop repetition condition tests each data item and causes loop exit when the sentinel value is read.
- Choose the sentinel value carefully; it must be a value that could not normally occur as data.

Syntax:

1.Get a line of data.

2. while the sentinel value has not been encountered

3. Process the data line.

4. Get another line of data.

For program readability, we usually name the sentinel by defining a constant macro.

Example Program

```
/* Compute the sum of a list of exam scores. */
#include<stdio.h>
#define SENTINEL -99
int main(void) {
int sum = 0, score;
printf("Enter first score (or %d to quit)> ", SENTINEL);
scanf("%d", &score);
while (score != SENTINEL) {
sum += score;
printf("Enter next score (%d to quit)> ", SENTINEL);
scanf("%d", &score);
}
printf("\nSum of exam scores is %d\n", sum);
return (0);
}
```

Endfile-Controlled Loops

A data file is always terminated by an endfile character that can be detected by the scanf function.

Therefore a batch program can be written that processes a list of data of any length without requiring a special sentinel value at the end of the data.

We have to set up a input loop so it notices when ***scanf*** encounters the ***endfile*** character.

scanf also returns a result value just like other functions.

When scanf is successfully able to fill its argument variables with values from the standard input device, the result value that it returns is the number of data items it actually obtained.

scanf function returns as its value the number of data items scanned.

Illustration:

```
int input_status
input_status = scanf("%d%d%lf", &part_id, &num_avail, &cost);
Printf(“The value returned by scanf is %d”, input_status );
```