

Repetition (Loop) Constructs: the `while` Statement

(C has other loop statements we'll examine later)

```
while (condition continues to be true)
/* if the condition is false, fall out of the loop */
{
    do this stuff,
    then go back to test the condition again
}
```

Logically, *every* loop must have 4 parts:

- Loop initialization statement(s)
- Loop termination condition
- Loop body (the stuff that we want to repeat)
- Loop modification statement(s)
- Programming languages do *not* usually force you to include these parts, so you must be careful to do it yourself. Otherwise your program will get wrong results.

```
#include <stdio.h>
int main()
{
    int Counter;                /* PRINT INTEGERS FROM 1 TO 10 */

    Counter = 1;                /* INITIALIZATION */

    while (Counter < 10)       /* TERMINATION COND. */
    {
        printf("%3d", Counter);
        Counter = Counter + 1; /* MODIFICATION */
    }
    printf("\n");
    return 0;
}
```

```

#include <stdio.h>
int main()
{
    int Outer;          /* counter for outer loop */
    int Inner;          /* counter for inner loop */

    Outer = 1;          /* INITIALIZATION */
    while (Outer < 7)   /* TERMINATION COND. */
    {
        printf("%3d  |", Outer);

        Inner = Outer; /* inner loop initialization */
        while (Inner <= 10) /* inner loop termination cond. */
        {
            printf("%3d  ", Inner);
            Inner = Inner + 1; /* inner loop modification */
        }

        printf("\n");
        Outer = Outer + 1; /* MODIFICATION */
    }
    printf("\n");
    return 0;
}

```

Algorithms: Repetitive Control Structures (loops)

There are 3 main kinds of loops:

- count-controlled
 - sentinel-controlled
 - flag-controlled
-
- *count-controlled*: before starting the repetition, calculate (or query the user about) how many iterations there will be
 - *sentintel-controlled*: start, and continue, the repetition until a problem-specific value appears in the input data
 - *flag-controlled*: after each iteration, ask the user whether to quit or continue

Counter-controlled repetition

Loop repeated until counter reaches a certain value

Definite repetition: number of repetitions is known before the repetition begins; might be constant, but more likely it's read as the first input value.

Example: A class of N 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.

Algorithm:

Set total to zero

Set grade counter to one

Prompt user and input the number of grades, N .

While grade counter is less than or equal to N

 Prompt user and 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 N

Print the class average

Sentinel-controlled repetition

Indicates “end of data entry.”

Loop ends when sentinel value appears in the input

Sentinel value chosen so it cannot be confused with a regular input (such as -1 in this case)

Revised averaging algorithm using sentinel and "priming read":

Input the first grade (possibly the sentinel)

While the user has not as yet entered the sentinel

 Add this grade into the running total

 Add one to the grade counter

 Input the next grade (possibly the sentinel)

Flag-controlled repetition

Loop ends when user enters a "quit or continue" true/false flag

Revised averaging algorithm using quit-flag:

Set total to zero

Set grade counter to one

Set quit-flag to "false" (or F, or 0, or whatever you are using)

While quit-flag is false

 Prompt user and input the next grade

 Add the grade into the total

 Add one to the grade counter

 Prompt user and input quit-flag value

Set the class average to the total divided by grade counter

Print the class average

Unit Price of a Pizza

Problem

You and your college roommates frequently order a late-night pizza snack. There are many pizzerias in the area that deliver to dormitories. Because you are on a tight budget, you would like to know which size pizza is the best value.

Analysis

To find which pizza is the best value, we must be able to do a meaningful comparison of pizza costs. Let's use *unit price*.

Pizzas are sold by size (diameter), measured in inches, so we can define the unit price of a pizza as its price divided by its area (that is, dollars per square inch).

The pizza problem

Initial Algorithm for Solution to Pizza Problem

1. For each size of pizza, read in the pizza size and price and compute unit cost. Compare the unit cost just computed with the previous unit costs and save the size and price of the pizza whose unit cost is the smallest so far.
2. Display the size and price of the pizza with the smallest unit cost.

The purpose of step 1 of the algorithm is to perform the cost computation for each individual pizza and somehow save the size and price of the pizza whose unit cost was the smallest. After all costs are computed, step 2 displays the size and price of the pizza that is the best buy.

Step 1 Refinement

- 1 *Repeat* the following steps for each size of pizza:
 - 1.1. Read in the next pizza size and price.
 - 1.2. Compute the unit price.
 - 1.3. *If* the new unit price is the smallest one so far,
then save this pizza's size, price, and unit price.

Step 1 specifies the *repetition* of a group of steps, which we *repeat* as many times as necessary until all unit prices are computed.

Each time we compute a new unit price, step 1.3 compares it to the others, and the current pizza's size and price are saved *if* its unit price is smaller than any others computed so far. *If* the unit price is not the smallest so far, the current pizza's size and price are not saved.

Step 1.3 is a selection step because it selects between the two possible outcomes: (a) save the pizza's data and (b) do not save the pizza's data.

How can we initialize the minimum value?

- set it to 0 before beginning the repetition
- make the first pizza a special case: read its data *before* beginning the repetition; this is called a *priming read*

Let's generalize the pizza problem
so the range of values can be negative
(say, minimum and maximum temperatures)

How do we initialize the min and max?

- priming read?
- set the min to the *maximum possible* value; set the max to the *minimum possible* value.
- C provides `limits.h`: `INT_MIN`, `INT_MAX` etc.