What is a loop?

- A loop is to execute a set of instructions repeatedly until a particular condition is being satisfied .
- That is, you can execute particular statements more than once in a controlled fashion
- Statements are executed as long as some condition remains true

Two Types of Loops

count controlled loops

repeat a specified number of times

event-controlled loops

some condition within the loop body changes and this causes the repetetion to stop

While Statement

SYNTAX

```
while ( Expression )
{       .
          .                    /*loop body */
         .
}
```

NOTE: Loop body can be a single statement, a null statement, or a block.

Parts of a While Loop

- Every while loop will always contain three main elements:
  - Priming: initialize your variables.
  - Testing: test against some known condition.
  - Updating: update the variable that is tested.

Count-controlled Loop

```
int   count ;

count  =  4;                          /* initialize loop
   variable */

while (count > 0)                     /* test expression */
{
      printf(" %d \n ",count ) ;  /* repeated action */

      count -- ;                      /*update loop variable */
}
printf( "Done" ) ;
```

1. Priming

2. Test Condition

3. Update

Computing Sum

- If we want to compute $\sum_{i=1}^{100} i$, we need to go 1+2+3+...+100

```
#include <stdio.h>
int main(void) {
    int sum =0, i = 1;
    while (i <= 100) {
        sum = sum + i;
        i = i + 1;
    }
    printf("Sum is %d\n", sum);
    return 0;      }
```

Infinite loop

- **A loop that never ends.**

- **Generally, you want to avoid these!**

- **There are special cases, however, when you do want to create infinite loops on purpose.**

Infinite While Loop

```
#include <stdio.h>
#define MAX 10
main ()

{
        int index =1;

        while (index <= MAX)
        {

                printf ("Index:  %d\n", index);

        }
}
```

1. Priming

2. Test Condition

3. Where is the update part

```
Index:   1
Index:   1
Index:   1
Index:   1
Index:   1

... [forever]
```

Infinite While Loop

```
#include <stdio.h>
/* no MAX here  */
main ()

{
        int index =1;

        while (index > 0)
        {

                printf ("Index:  %d\n", index);
                index = index + 1;

        }
}
```

1. Priming

2. Test Condition

3. Update

```
Index:  1
Index:  2
Index:  3
Index:  4
Index:  5


... [forever]
```

Event controlled loop

- Signals the end of data entry
- Also known as signal value, dummy value, or flag value
- Also known as indefinite repetition because the number of repetitions the code will perform is unknown before the loop

Event controlled loop
Flag-controlled Loops

- How are they used?
  - Programmer picks a value that would never be encountered for normal data
  - User enters normal data and then when done, enters the unusual value
  - The loop would stop when seeing the unusual value

Do-While Statement

Is a looping control structure in which the loop condition is tested after each iteration of the loop.

SYNTAX

```
do
{
        Statements

} while ( Expression ) ;
```

Loop body statement can be a single statement or a block.

Computing Sum

- If we want to compute $\sum_{i=1}^{100} i$, we need to go 1+2+3+...+100

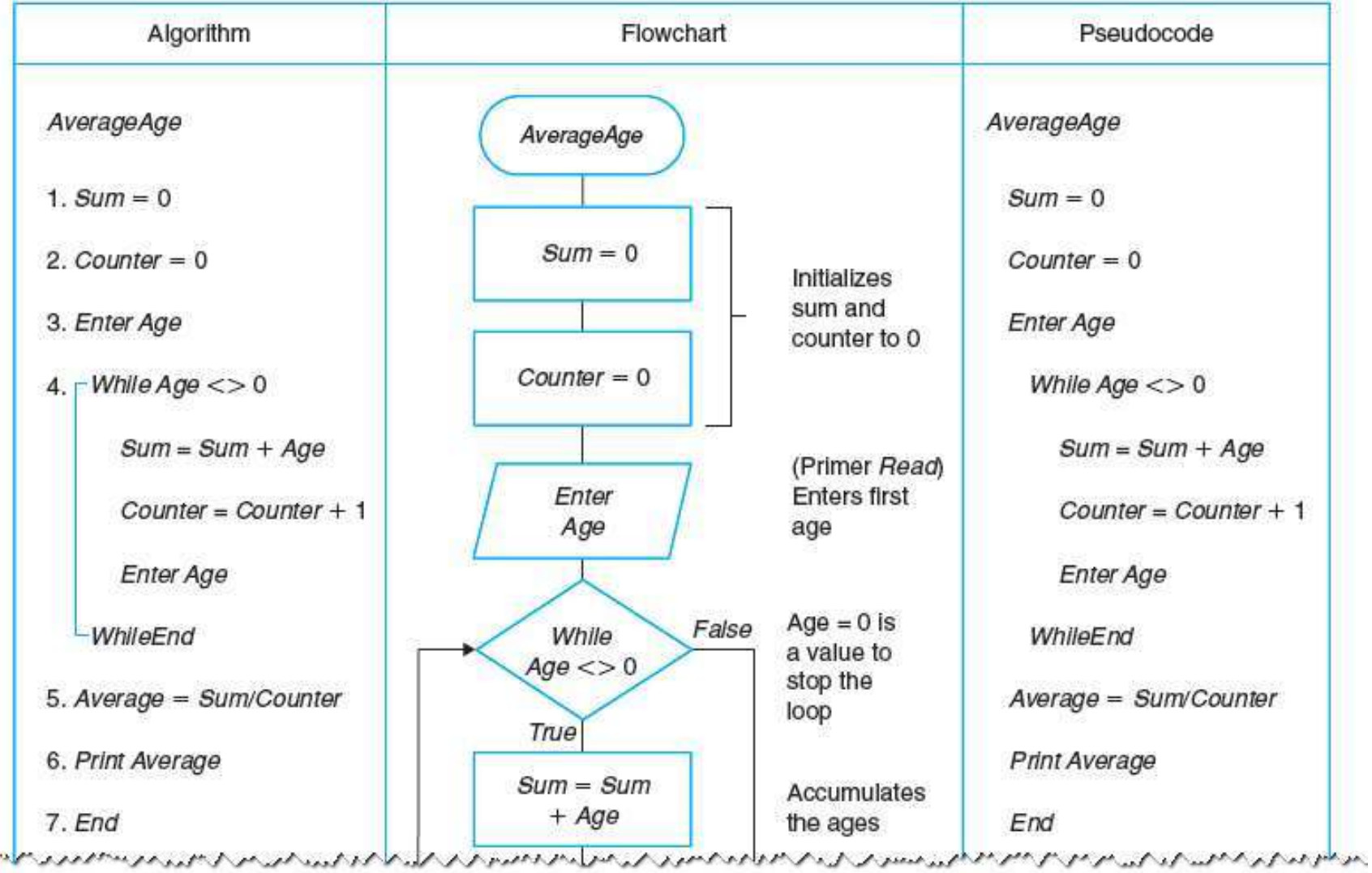
```
/* computes the sum: 1 + 2 + 3 + ....+ 100 */
#include <stdio.h>
int main(void)
{
.....
......
......

    printf("Sum is %d\n", sum);
    return 0;      }
```

Do-While Loop vs. While Loop

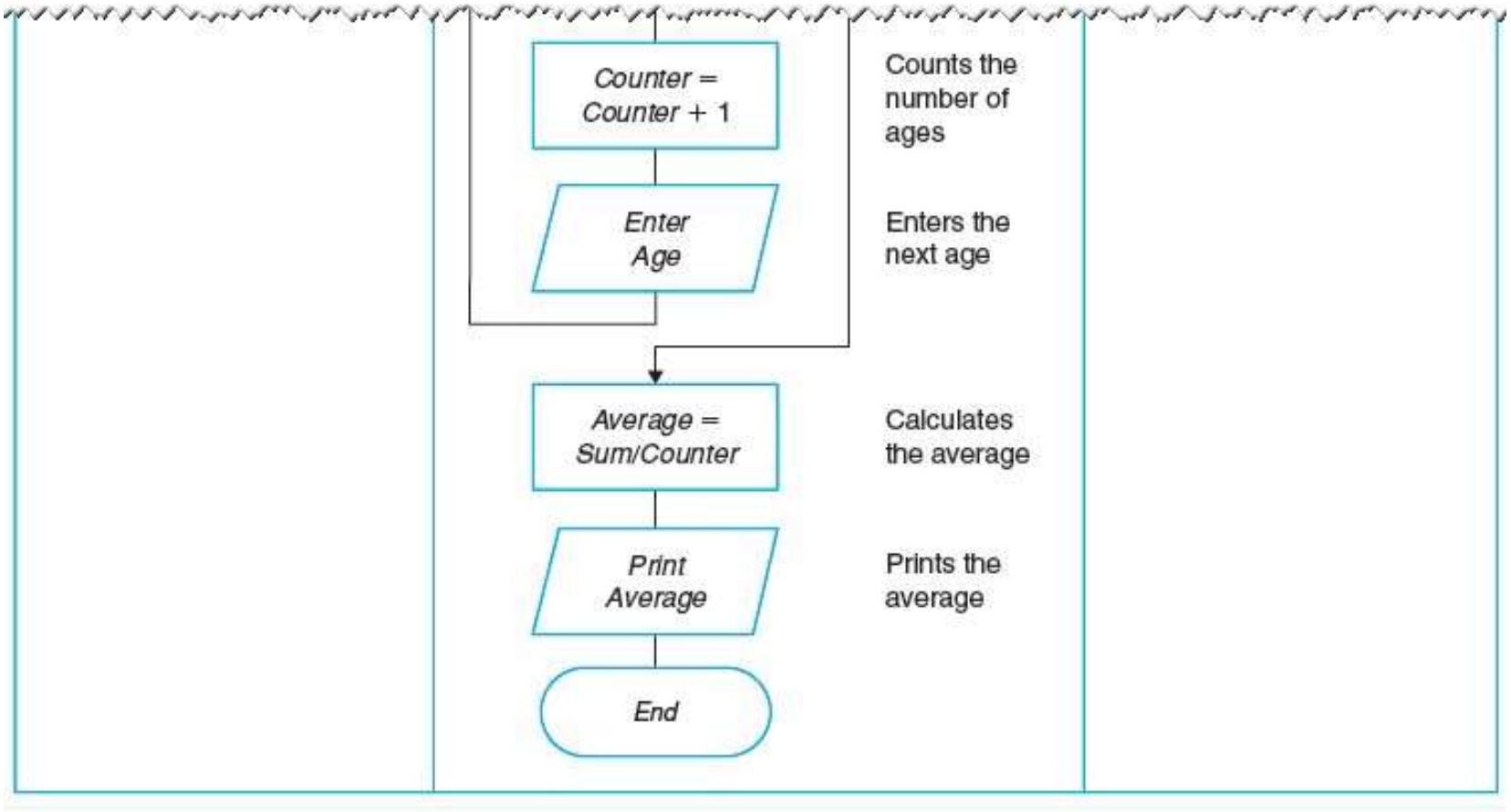
- POST-TEST loop (exit-condition)
- The looping condition is tested after executing the loop body.
- Loop body is always executed at least once.

- PRE-TEST loop (entry-condition)
- The looping condition is tested before executing the loop body.
- Loop body may not be executed at all.

Example: Average Age of a Class — *While/WhileEnd*



Example: Average Age of a Class — *While/WhileEnd*



Basic For Loop Syntax

- For loops are good for creating definite loops.

```
int counter;

for (counter =1;counter <= 10;counter++)
printf ("%d\n", counter);
```

1. Priming: Set the start value.

2. Test Condition: Set the stop value.

3. Update: Update the value.

Note that each section is separated by a semicolon.

Computing Sum

- If we want to compute $\sum_{i=1}^{100} i$, we need to go 1+2+3+...+100

```
/* computes the sum: 1 + 2 + 3 + ....+ 100 */
#include <stdio.h>
int main(void)
{
……..

……..

printf("Sum is %d\n", sum);
return 0; }
```