`while` Loop Outline

1. `while` Loop Outline
2. `while` Loop Example #1
3. `while` Loop Example #2
4. `while` Loop Example #3
5. Repetition and Looping
6. `while` Loop
7. `while` Loop Behavior
8. `while` Loop vs. `if` Block
9. `while` Loop Flowchart
10. `while` Loop Example #1
11. `while` Loop Example #2
12. `while` Loop Example #3
13. `while` Loop Example Flowchart
14. Execute Body How Many Times?
15. An Infinite Loop #1
16. An Infinite Loop #2
17. Aside: How to Kill a Program in Unix
18. Kinds of Statements Inside `while` Loop
19. Statements Inside `while` Loop
20. No Declarations Inside `while` Loop
21. Compound Statement a.k.a. Block #1
22. Compound Statement a.k.a. Block #2
23. Another `while` Loop Example #1
24. Another `while` Loop Example #2
25. Another `while` Loop Example #3
26. Another `while` Loop Example #4
27. Another `while` Loop Example #5
28. Yet Another `while` Loop Example #1
29. Yet Another `while` Loop Example #2
30. Yet Another `while` Loop Example #3
31. Yet Another `while` Loop Example #4
32. States & Traces #1
33. States & Traces #2
34. States & Traces #3
35. Tracing the Loop #1
36. Tracing the Loop #2
37. Tracing the Loop #3
38. Tracing the Loop #4
39. Tracing the Loop #5

while Loop Example #1

```
#include <stdio.h>
#include <stdlib.h>

int main ()
{ /* main */
    const float minimum_volume        =  0;
    const int   program_success_code =  0;
    const int   program_failure_code = -1;
    float volume_in_fluid_ounces;
```

while Loop Example #2

```
    printf("What is the volume in fluid ounces?\n");
    scanf("%f", &volume_in_fluid_ounces);
    while (volume_in_fluid_ounces <
           minimum_volume) {
        printf("ERROR: you can't have a");
        printf(" negative volume %f!\n",
            volume_in_fluid_ounces);
        printf("So really, what is the");
        printf(" volume in fluid ounces?\n");
        scanf("%f", &volume_in_fluid_ounces);
    } /* while (volume_in_fluid_ounces < ...) */
    printf("The volume in fluid ounces is valid.\n");
    return program_success_code;
} /* main */
```

while Loop Example #3

```
% gcc -o volume_idiot_while volume_idiot_while.c
% volume_idiot_while
What is the volume in fluid ounces?
-5
ERROR: you can't have a negative volume -5.00000!
So really, what is the volume in fluid ounces?
-4
ERROR: you can't have a negative volume -4.00000!
So really, what is the volume in fluid ounces?
0
The volume in fluid ounces is valid.
```

Repetition and Looping

Repetition means performing the same set of statements over and over.

The most common way to perform repetition is via ***looping***.

A ***loop*** is a sequence of statements to be executed, in order, over and over, as long as some condition continues to be true.

`while` Loop

C has a loop construct known as a `while` loop:

```
while (condition) {
    statement1;
    statement2;
    ...
}
```

The condition of a `while` loop is
a Boolean expression completely enclosed in parentheses – just like the condition of an `if` block.

The sequence of statements between the `while` statement's block open and block close is known as the ***loop body***.

`while` Loop Behavior

```
while (condition) {
    statement1;
    statement2;
    ...
}
```

A `while` loop has to the following behavior:

1. The condition is evaluated, resulting in a value of either true (`1`) or false (`0`).
2. If the condition evaluates to false (`0`), then the statements inside the loop body are skipped, and control is passed to the statement that is **IMMEDIATELY AFTER** the `while` loop's block close.
3. If the condition evaluates to true (`1`), then:
   a. the statements inside the loop body are executed in order.
   b. When the `while` loop's block close is encountered, the program jumps back up to the associated `while` statement and starts over with Step 1.

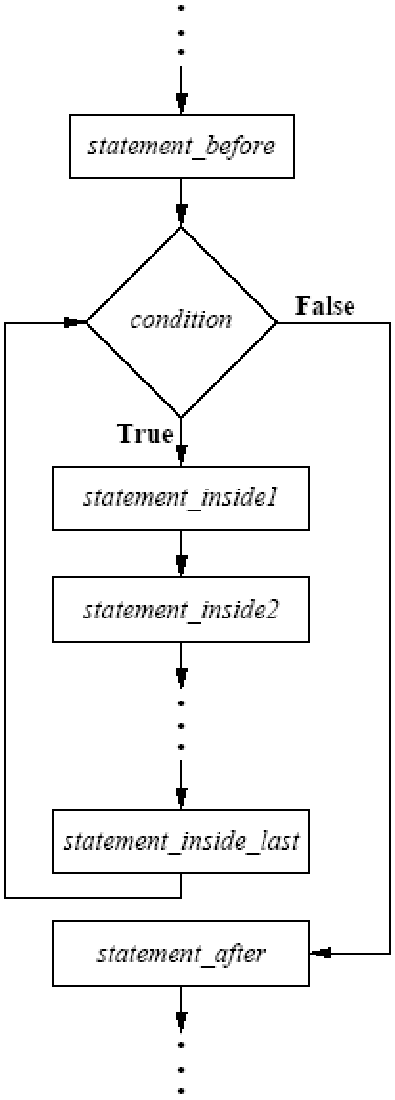
`while` Loop vs. `if` Block

A **`while` loop** is **SIMILAR** to an **`if` block**, **EXCEPT**:

1. **UNLIKE** an `if` block, the **keyword** is `while`.
2. **UNLIKE** an `if` block, when a `while` loop gets to its block close, it **jumps back up** to the associated `while` statement.
3. **UNLIKE** an `if` block,
   a `while` loop has **EXACTLY ONE** clause,
   which is **analogous to the `if` clause.**
   A `while` loop **CANNOT** have anything analogous to an `else if` clause nor to an `else` clause.

`while` Loop Flowchart

```
statement_before;
while (condition) {
      statement_inside1;
      statement_inside2;
      ...
}
statement_after;
```



while Loop Example #1

```
#include <stdio.h>
#include <stdlib.h>

int main ()
{ /* main */
    const float minimum_volume =  0;
    const int   program_success_code   =  0;
    const int   program_failure_code   = -1;
    float volume_in_fluid_ounces;
```

`while` Loop Example #2

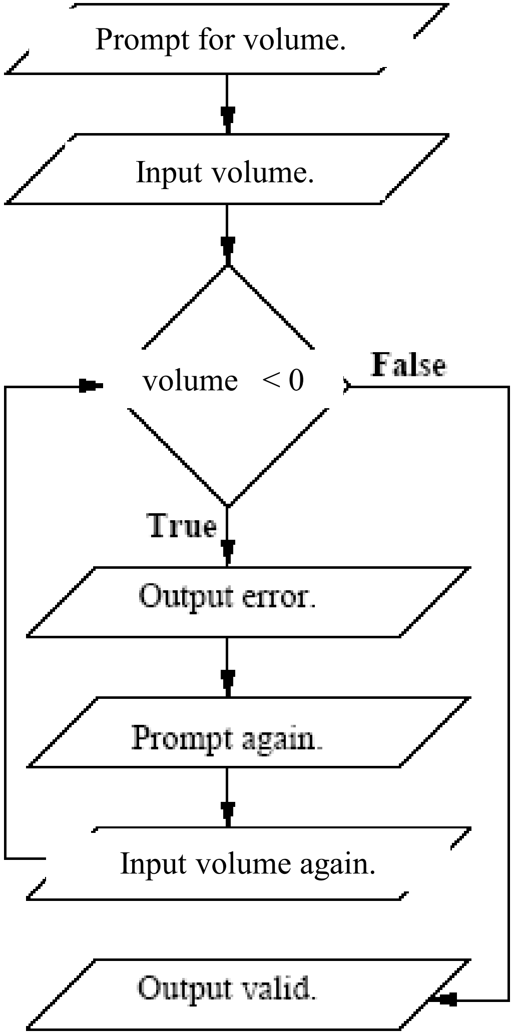
```
    printf("What is the volume in fluid ounces?\n");
    scanf("%f", &volume_in_fluid_ounces);
    while (volume_in_fluid_ounces <
           minimum_volume) {
        printf("ERROR: you can't have a");
        printf(" negative volume!\n");
        printf("So really, what is the ");
        printf(" volume in fluid ounces?\n");
        scanf("%f", &volume_in_fluid_ounces);
    } /* while (volume_in_fluid_ounces < ...) */
    printf("The volume in fluid ounces is valid.\n");
    return program_success_code;
} /* main */
```

while Loop Example #3

```
% gcc -o volume_idiot_while volume_idiot_while.c
% volume_idiot_while
What is the volume in fluid ounces?
-5
ERROR: you can't have a negative volume!
So really, what is the volume in fluid ounces?
-4
ERROR: you can't have a negative volume!
So really, what is the volume in fluid ounces?
0
The volume in fluid ounces is valid.
```

`while` Loop Example Flowchart

```
printf("What is the volume in fluid ounces?\n");
scanf("%f", &volume_in_fluid_ounces);
while (volume_in_fluid_ounces <
       minimum_volume) {
    printf("ERROR: you can't have a");
    printf(" negative volume!\n");
    printf("So really, what is the");
    printf(" volume in fluid ounces?\n");
    scanf("%f", &volume_in_fluid_ounces);
} /* while (volume_in_fluid_ounces < ...) */
printf("the volume in fluid ounces is valid.\n");
```



Execute Body How Many Times?

```
while (condition) {
        statement1;
        statement2;
        ...
}
```

If the condition evaluates to false (`0`), then
  the loop body won't be executed at all (that is, zero times).

If the condition evaluates to true (`1`), then
  the loop body might be executed at least one more time.

An Infinite Loop #1

An ***infinite loop*** is a loop whose condition **NEVER** evaluates to false.

```
#include <stdio.h>

int main ()
{ /* main */
    const int computers_number     = 5;
    const int program_success_code = 0;
    int users_number;

    printf("Enter an integer:\n");
    scanf("%d", &users_number);
    printf("I had %d.\n", computers_number);
    while (users_number < computers_number) {
        printf("Your number is less than mine!\n");
    } /* while (users_number < computers_number) */
    return program_success_code;
} /* main */
```

An Infinite Loop #2

```
% gcc -o infiniteloop infiniteloop.c
% infiniteloop
Enter an integer:
6
I had 5.
% infiniteloop
Enter an integer:
5
I had 5.
% infiniteloop
Enter an integer:
4
I had 5.
Your number is less than mine!
Your number is less than mine!
Your number is less than mine!
Your number is less than mine!
Your number is less than mine!
Your number is less than mine!
Your number is less than mine!
Your number is less than mine!
...
```

Aside: How to Kill a Program in Unix

On most Unix systems, including `ssh.ou.edu`, you can quit out of a program that is currently executing by typing:

Ctrl - `C`

Kinds of Statements Inside `while` Loop

Between the `while` statement's block open and its associated block close, there can be **<u>any kind</u>** of **<u>executable</u>** statements, and **<u>any number</u>** of them.

For example:

- `printf` statements;
- `scanf` statements;
- assignment statements;
- `if` blocks;
- `while` loops.

There are several other kinds of executable statements that can occur inside a `while` loop, some of which we'll learn later in the semester.

Statements Inside `while` Loop

In the event that the `while` condition evaluates to true (`1`), then the statements inside the `while` loop body will be executed one by one, in the order in which they appear in the `while` loop.

No Declarations Inside `while` Loop

Notice that a `while` loop
SHOULDN'T contain declaration statements,
because the `while` statement is an executable statement,
and **ALL** declarations **MUST** come
before **ANY** executable statements.

Compound Statement a.k.a. Block #1

A ***compound statement*** is a sequence of statements, with a well-defined beginning and a well-defined end, to be executed, in order, under certain circumstances.

A `while` loop is a compound statement, just like an `if` block. We'll see others later.

Although a `while` loop is actually a sequence of statements, we can treat it as a single "super" statement in some contexts.

Compound statements are also known as ***blocks***.

Compound Statement a.k.a. Block #2

In C, a compound statement, also known as a block, is delimited by curly braces.

That is, a compound statement/block begins with a block open

```
{
```

and ends with a block close

```
}
```

Another while Loop Example #1

```
#include <stdio.h>

int main ()
{ /* main */
    const int false                =   0;
    const int true                 =   1;
    const int minimum_number       =   1;
    const int maximum_number       = 100;
    const int computers_number     =  32;
    const int close_distance       =   1;
    const int negative_distance    =  -1;
    const int no_distance          =   0;
    const int program_success_code =   0;
    int  users_number, users_distance;
    int  users_last_distance = negative_distance;
    char correct_number_hasnt_been_input = true;
```

Another `while` Loop Example #2

```
printf("I'm thinking of a number between %d and %d.\n",
    minimum_number, maximum_number);
while (correct_number_hasnt_been_input) {
    printf("What number am I thinking of?\n");
    scanf("%d", &users_number);
    if ((users_number < minimum_number) ||
        (users_number > maximum_number)) {
        printf("Hey! That's not between %d and %d!\n",
            minimum_number, maximum_number);
        printf("I'll pretend you didn’t say that.\n");
    } /* if ((users_number < minimum_number) || ...) */
    else if (users_number == computers_number) {
        printf("That's amazing!\n");
        correct_number_hasnt_been_input = false;
    } /* if (users_number == computers_number) */
```

Another `while` Loop Example #3

```
        else {
            users_distance =
                abs(users_number - computers_number);
            if (users_distance == close_distance) {
                printf("You're incredibly hot!\n");
            } /* if (users_distance == close_distance) */
            else if (users_last_distance < no_distance) {
                printf("Not bad for your first try.\n");
            } /* if (users_last_distance < no_distance) */
            else if (users_distance < users_last_distance) {
                printf("You're getting warmer ....\n");
            } /* if (users_distance < users_last_distance) */
            else if (users_distance > users_last_distance) {
                printf("Ouch! You’re getting colder.\n");
            } /* if (users_distance > users_last_distance) */
            else {
                printf("Uh oh. You made no progress.\n");
            } /* if (users_distance > ...)...else */
            users_last_distance = users_distance;
        } /* if (users_number == computers_number)...else */
    } /* while (correct_number_hasnt_been_input) */
    printf("Good for you!\n");
    return program_success_code;
} /* main */
```

Another while Loop Example #4

```
% gcc -o warmercolder warmercolder.c
% warmercolder
I'm thinking of a number between 1 and 100.
What number am I thinking of?
0
Hey! That's not between 1 and 100!
I'll pretend you didn’t say that.
What number am I thinking of?
101
Hey! That's not between 1 and 100!
I'll pretend you didn’t say that.
What number am I thinking of?
50
Not bad for your first try.
What number am I thinking of?
40
You're getting warmer ....
What number am I thinking of?
60
Ouch! You’re getting colder.
```

Another while Loop Example #5

```
What number am I thinking of?
30
You're getting warmer ....
What number am I thinking of?
35
Ouch! You're getting colder.
What number am I thinking of?
33
You're incredibly hot!
What number am I thinking of?
31
You're incredibly hot!
What number am I thinking of?
32
That's amazing!
Good for you!
```

Yet Another while Loop Example #1

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{ /* main */
    const int initial_sum          =  0;
    const int increment            =  1;
    const int program_success_code =  0;
    const int program_failure_code = -1;
    int initial_value, final_value;
    int count;
    int sum;
```

Yet Another `while` Loop Example #2

```
printf("What value would you like to ");
printf("start counting at?\n");
scanf("%d", &initial_value);
printf("What value would you like to ");
printf("stop counting at,\n");
printf("  which must be greater than ");
printf("or equal to %d?\n", initial_value);
scanf("%d", &final_value);
if (final_value < initial_value) {
    printf("ERROR: the final value %d is less\n",
        final_value);
    printf("  than the initial value %d.\n",
        initial_value);
    exit(program_failure_code);
} /* if (final_value < initial_value) */
```

Yet Another `while` Loop Example #3

```
    sum   = initial_sum;
    count = initial_value;
    while (count <= final_value) {
        sum = sum + count;
        count = count + increment;
    } /* while (count <= final_value) */
    printf("The sum of the integers from");
    printf(" %d through %d is %d.\n",
        initial_value, final_value, sum);
    return program_success_code;
} /* main */
```

Yet Another while Loop Example #4

```
% gcc -o whilecount whilecount.c
% whilecount
What value would you like to start counting at?
1
What value would you like to stop counting at,
  which must be greater than or equal to 1?
0
ERROR: the final value 0 is less
  than the initial value 1.
% whilecount
What value would you like to start counting at?
1
What value would you like to stop counting at,
which must be greater than or equal to 1?
5
The sum of the integers from 1 through 5 is 15.
```

States & Traces #1

The ***state*** of a program is the set of values of all of its variables at a given moment during execution; that is, it's a **snapshot** of the memory that's being used.

The state also includes information about **where you are** in the program when that snapshot is taken.

A ***trace*** of a program is a listing of the state of the program after each statement is executed.

Tracing helps us to examine the behavior of a piece of code, so it sometimes can be useful in debugging.

States & Traces #2

Suppose that, in the previous example program, the user input 1 for `initial_value` and 5 for `final_value`.

Let’s examine the program fragment around the loop.

```
sum   = initial_sum;
count = initial_value;
while (count <= final_value) {
    sum = sum + count;
    count = count + increment;
} /* while (count <= final_value) */
```

States & Traces #3

```
sum   = initial_sum;
count = initial_value;
while (count <= final_value) {
    sum = sum + count;
    count = count + increment;
} /* while (count <= final_value) */
```

If we number these statements, we get:

```
1   sum   = initial_sum;
2   count = initial_value;
3   while (count <= final_value) {
4       sum = sum + count;
5       count = count + increment;
6   } /* while (count <= final_value) */
```

Tracing the Loop #1

```
1   sum   = initial_sum;
2   count = initial_value;
3   while (count <= final_value) {
4       sum = sum + count;
5       count = count + increment;
6   } /* while (count <= final_value) */
```

Snapshot of		Trace		Comments
Iteration #	After stmt #	Value of `sum`	Value of `count`	
N/A	1	0	**garbage**	Haven’t entered loop yet
N/A	2	0	1	Haven’t entered loop yet
1	3	0	1	Condition evaluates to true (1)
1	4	1	1	new `sum` = old `sum` + `count` = 0 + 1 = 1
1	5	1	2	new `count` = old `count` + 1 = 1 + 1 = 2
1	6	1	2	Jump back up to stmt #3 to start iteration #2

Tracing the Loop #2

```
1   sum   = initial_sum;
2   count = initial_value;
3   while (count <= final_value) {
4       sum = sum + count;
5       count = count + increment;
6   } /* while (count <= final_value) */
```

Snapshot of		Trace		Comments
Iteration #	After stmt #	Value of `sum`	Value of `count`	
2	3	1	2	Condition evaluates to true (`1`)
2	4	3	2	new `sum` = old `sum` + `count` = 1 + 2 = 3
2	5	3	3	new `count` = old `count` + 1 = 2 + 1 = 3
2	6	3	3	Jump back up to stmt #3 to start iteration #3

Tracing the Loop #3

```
1   sum   = initial_sum;
2   count = initial_value;
3   while (count <= final_value) {
4       sum = sum + count;
5       count = count + increment;
6   } /* while (count <= final_value) */
```

Snapshot of		Trace		Comments
Iteration #	After stmt #	Value of `sum`	Value of `count`	
3	3	3	3	Condition evaluates to true (`1`)
3	4	6	3	new `sum` = old `sum` + `count` = `3` + `3` = `6`
3	5	6	4	new `count` = old `count` + `1` = `3` + `1` = `3`
3	6	6	4	Jump back up to stmt #3 to start iteration #4

Tracing the Loop #4

```
1   sum   = initial_sum;
2   count = initial_value;
3   while (count <= final_value) {
4       sum = sum + count;
5       count = count + increment;
6   } /* while (count <= final_value) */
```

Snapshot of		Trace		Comments
Iteration #	After stmt #	Value of `sum`	Value of `count`	
4	3	6	4	Condition evaluates to true (`1`)
4	4	10	4	new `sum` = old `sum` + `count` = 6 + 4 = 10
4	5	10	5	new `count` = old `count` + 1 = 4 + 1 = 5
4	6	10	5	Jump back up to stmt #3 to start iteration #5

Tracing the Loop #5

```
1   sum   = initial_sum;
2   count = initial_value;
3   while (count <= final_value) {
4       sum = sum + count;
5       count = count + increment;
6   } /* while (count <= final_value) */
```

Snapshot of		Trace		Comments
Iteration #	After stmt #	Value of `sum`	Value of `count`	
5	3	10	5	Condition evaluates to true (`1`)
5	4	15	5	new `sum` = old `sum` + `count` = `10` + `5` = `15`
5	5	15	6	new `count` = old `count` + `1` = `5` + `1` = `6`
5	6	15	6	Jump back up to stmt #3 to start iteration #6
5	6	15	6	Condition evaluates to false (`0`), loop exited