UESTC 1005 – Introductory Programming

Lecture 4 – Loops

Dr Hasan T Abbas
Hasan.Abbas@glasgow.ac.uk

Fall 2019

Glasgow College – UESTC

Iteration Explained

- An iteration statement lets us perform an action repeatedly while a certain condition remains true. A loop is a group of instructions a computer executes repeatedly

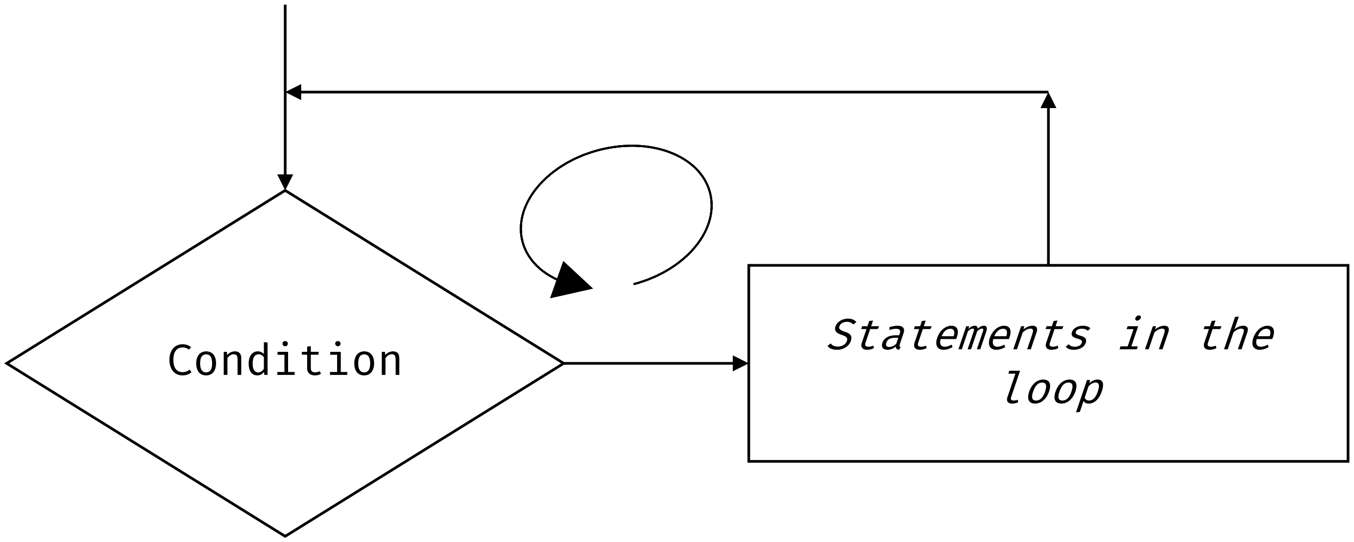
  Example Pseudocode:

  ```
  While I am in IP student
      Come to class on Mon, Tue and Fri on Weeks 3,8,10,14
  ```

- In the above case, the condition is <u>being an IP student</u> (Fall semester 2019).
- If it is true, then the student will come to the class on the days specified.
- The task will be performed until the end of the semester

Loops Explained

- A loop is a group of instructions a computer executes repeatedly
- There is always a condition that stays TRUE while the loop is repeated.
- We call each execution of a loop as as iteration
- In C, every loop has a controlling expression



Loops in C programming

In C, we have three types of loop statements:

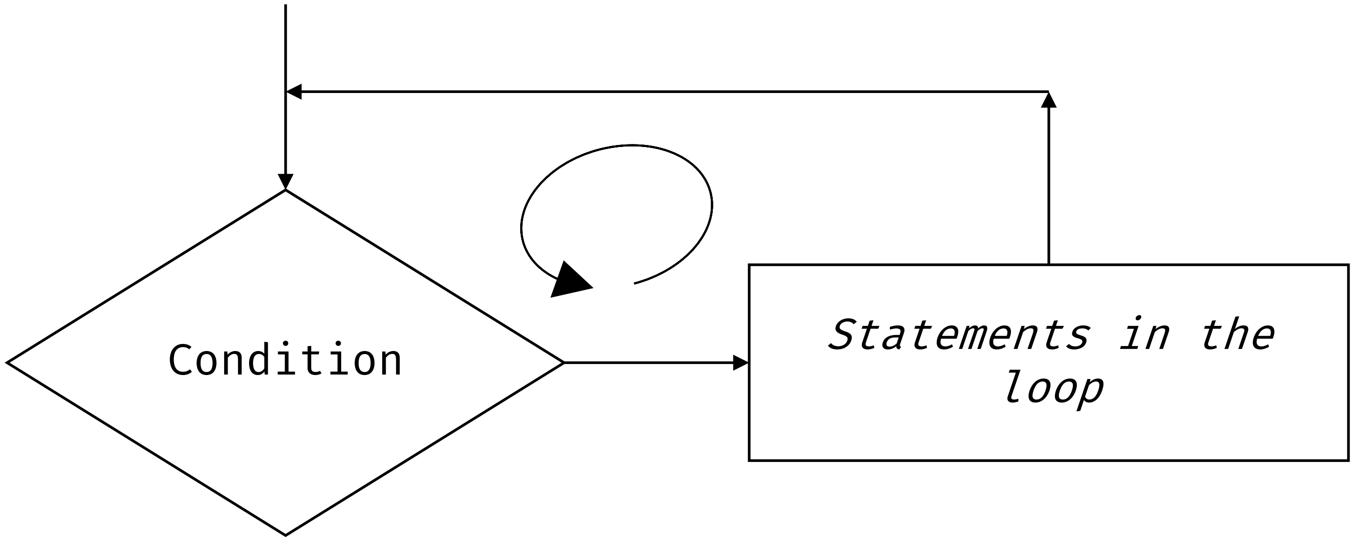
1. `while`
2 `do-while`
3. `for`

- Jump statements (`break` and `continue`) are also important in implementing the loops in C programming.
- The `while` statement is used for loops in which the controlling expression is tested ***before*** the loop body.
- `do-while` is used in which the controlling expression is executed ***after*** the loop body
- In a `for` loop, we increment or decrement a counting variable

The `while` loop

Simplest and most fundamental

```
while (controlling expression){
    expression statements;
}
```



The `while` loop - example

We check the condition first and then execute the loop body.

```
while (a < 2){
    a++;
    printf("a is %d", a);
}
```

```
int main(){
int i = 1;
int n = 20;
while (i < n) // controlling expression
{
   i = i * 2;
  printf("i is %d\n", i);
}
Output:
i is 2
i is 4
i is 8
i is 16
i is 32
```

The `while` loop – step by step explanation

```
int main(){
int x = 1;
int n = 20;
while (x < n) // controlling expression
{
   x = x * 2;
  printf("x is %d\n", x);
}
Output:
x is 2
x is 4
x is 8
x is 16
x is 32
```

1. Is 1 < 20 (true)

   x = 1 * 2 = 2
2. Is 2 < 20 (true)

   x = 2 *2 = 4
3. Is 4 < 20 (true)

   x = 4 *2 = 8
4. Is 8 < 20 (true)

   x = 8 * 2 = 16
5. Is 16 < 20 (true)

   x = 16 * 2 = 32
6. Is 32 < 20 (false)

The `while` loop - another example

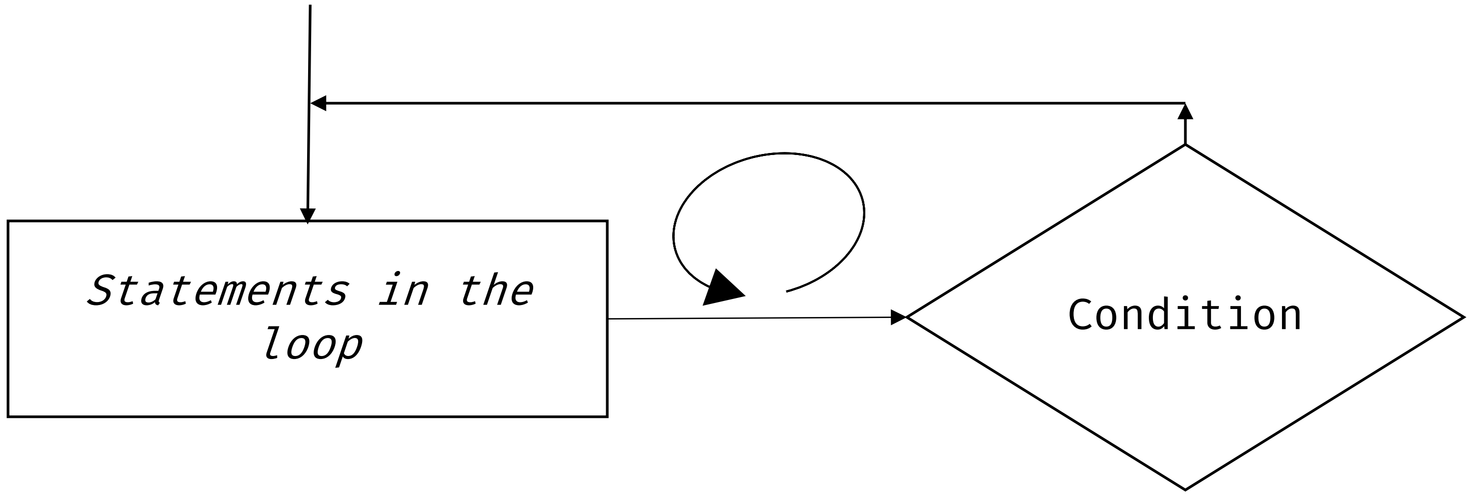
```
int main(){
int i = 1;
int n = 3;
while (i <= n) {
    printf("%10d %10d\n", i, i * i);
    i++;
  }


Output:
```

The `do-while` loop

Essentially the same as a while loop

```
do {
   expression statements;
} while (controlling expression); // notice the semicolon
```



The `do-while` loop - example

We perform the loop statements first and then the condition is checked

```
int main(){
int i = 1;
int n = 20;
do {
   i = i * 2;
  printf("i is %d\n", i);
} while (i < n) // controlling expression

Output:
i is 2
i is 4
i is 8
i is 16
```

The `do-while` loop – step by step explanation

```
int main(){
int i = 1;
int n = 20;
do {
   i = i * 2;
  printf(“i is %d\n”, i);
} while (i < n) // controlling
expression

Output:
i is 2
i is 4
i is 8
i is 16
```

x = 1 * 2 = 2

Is 2 < 20 (true)

x = 2 *2 = 4

Is 4 < 20 (true)

x = 4 *2 = 8

Is 8 < 20 (true)

x = 8 * 2 = 16

Is 16 < 20 (true)

x = 16 * 2 = 32

Is 32 < 20 (false)

The `while` loop - another example

```
 int digits = 0, n;

  printf("Enter a nonnegative integer: ");
  scanf("%d", &n);

  do {
    n /= 10;
    digits++;
  } while (n > 0);

  printf("The number has %d digit(s).\n", digits);

Output:
```

The `for` loop

You will use it most often in your programs

```
for (expr1 ; expr2 ; expr3){
  expression statements;
}
```

Example

```
      1        2        4
for (j = 0; j <= 10 ; j++){
  printf("Counting numbers %d \n", j); 3
}
```

for loop is very similar to the while loop

The `for` loop - example

```
int main()
{
    int n, counter = 0, value = 1;
    scanf("%d", &n);

    for (counter=0; counter<=n; counter++)
    {
        if (n == 0)
            printf("value is 1\n");
        else
        {
            value *= 2;
            printf("value is %d\n", value);
        }
    }
}
```

for loop body

The `for` loop - examples

1. Vary the control variable from 1 to 100 in increments of 1.

```
for ( i = 1; i <= 100; i++ )
```

2. Vary the control variable from 100 to 1 in increments of -1 (decrements of 1).

```
for ( i = 100; i >= 1; i-- )
```

3. Vary the control variable from 7 to 77 in steps of 7.

```
for ( i = 7; i <= 77; i += 7 )
```

4. Vary the control variable from 20 to 2 in steps of -2.

```
for ( i = 20; i >= 2; i -= 2 )
```

5. Vary the control variable over the following sequence of values: 2, 5, 8, 11, 14, 17.

```
for ( j = 2; j <= 17; j += 3 )
```

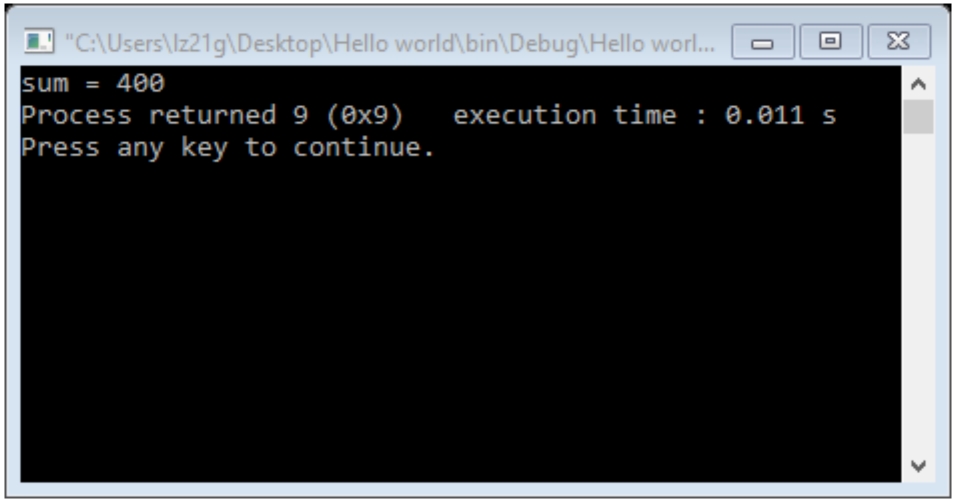
6. Vary the control variable over the following sequence of values: 44, 33, 22, 11, 0.

```
for ( j = 44; j >= 0; j -= 11 )
```

Example - Sum of the first 20 odd numbers

```c
int main()
{
    int i, j = 1, sum =0;

    for (i=1; i<=20; i++)
    {
        sum += j;
        j += 2;
    }
    printf("sum = %d", sum);
}
```

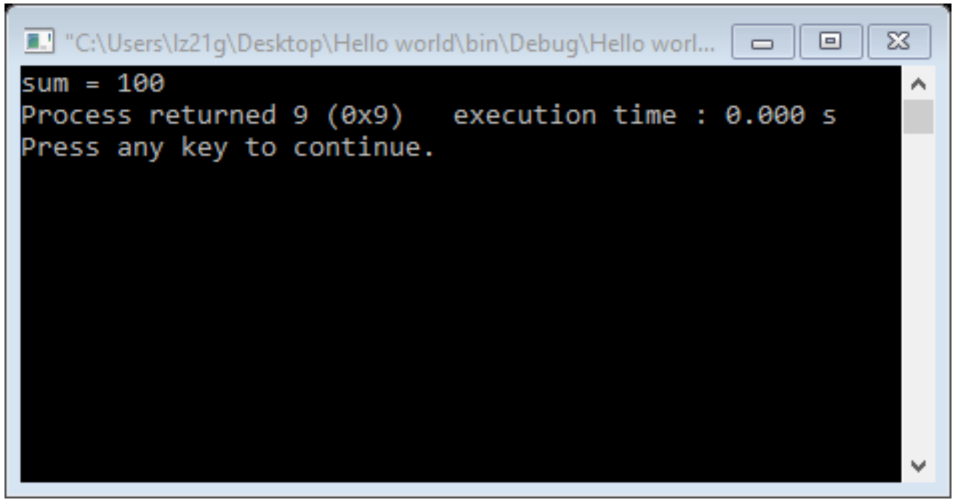


```
sum = 400
Process returned 9 (0x9)   execution time : 0.011 s
Press any key to continue.
```

Example - Sum the odd numbers between 1 to 20

```c
int main()
{
    int i, sum =0;

    for (i=1; i<=20; i+=2)
    {
        sum += i;
        //i += 2;
    }
    printf("sum = %d", sum);
}
```



```
sum = 100
Process returned 9 (0x9)   execution time : 0.000 s
Press any key to continue.
```

`break` and `continue` statements

- The break and continue statements are used to alter the flow of control.
- The break statement, when executed in a while, for, do…while or switch statement, causes an immediate exit from that statement.

```c
int main()
{
   int x;
   for ( x = 1; x <= 10; x++ ) {
      if ( x == 5 )
         break;    /* break loop only if x == 5 */
      printf( "%d ", x );
   }
   printf( "\nBroke out of loop at x == %d\n", x );
   return 0;
}
```

```
1 2 3 4
Broke out of loop at x == 5
```

`break` and `continue` statements

- The continue statement, when executed in a while, for or do…while statement, skips the remaining statements in the body of that control statement and performs the next iteration of the loop.

```
int main()
{
   int x;
   for ( x = 1; x <= 10; x++ ) {
      if ( x == 5 )
         continue;  /* skip remaining code in loop only
                       if x == 5 */
      printf( "%d ", x );
   }
   printf( "\nUsed continue to skip printing the value 5\n" );
   return 0;
}
```

```
1 2 3 4 6 7 8 9 10
Used continue to skip printing the value 5
```

Next Lecture …

Nested Loops

Functions