Computer Programming & Problem Solving
CS100

Mrs Sanga G. Chaki
Department of Computer Science and Engineering
National Institute of Technology, Goa
March, 2023

Some new terms

1. Counter – Some variable that counts the number of repetitions.
2. Iteration – the repetition of a process
3. Dry run – a rehearsal of a process before the real one.

Loops

1. Loops: Group of instructions that are executed repeatedly while some condition remains true.
2. While – entry controlled
3. For – entry controlled
4. Do-while – exit controlled
5. There are 4 steps that happen in a loop:
   a) Initialize counter – to count number of passes
   b) Check condition
   c) Go through/skip through loop - iteration
   d) Update counter

While Loop

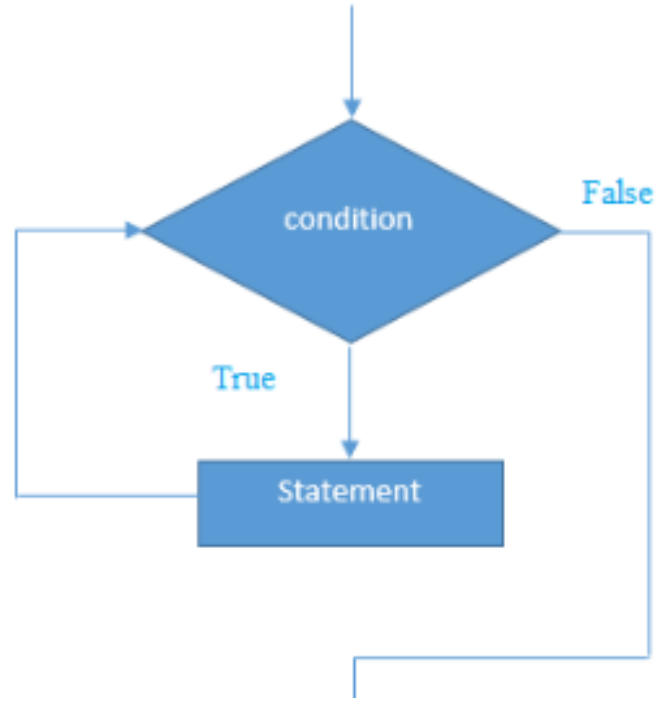
1. while(condition){

   statement_1;

   …

   statement_N; }

2. The while loop will not be entered if the loop control expression evaluates to false (zero) even before the first iteration.



While Loop: Sum of first N natural numbers

```
int main () {
    int N, count, sum;
    scanf ("%d", &N) ;
    sum = 0; count = 1;

    while (count <= N)  {
        sum = sum + count;
        count = count + 1;
    }

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

Increment/Decrement Operator

1. the increment operator ++ increments the value of i by 1, every time the statement i++ gets executed.
2. To reduce the value of a variable by 1 a decrement operator- - is used.
3. i=i+1
4. i++
5. i+=1 are all same
6. Post increment/pre increment

For Loop

1. General form
   a) for(expr1; expr2;expr3)
      - Statement;
2. expr1: initializes loop parameters
3. expr2: test condition, loop continues if this is satisfied
4. expr3: used to alter the value of the parameters after each iteration
5. statement: body of the loop

For Loop: Sum of first N natural numbers

```
int main () {
    int N, count, sum;
    scanf (“%d”, &N) ;

    sum = 0;
    for (count=1; count <= N; count++)
        sum = sum + count;

    printf (“Sum = %d\n”, sum) ;
    return 0;
}
```

Do While Loop

1. The do-while loop looks like this:

```
do
{
    this ;
    and this ;
    and this ;
    and this ;
} while ( this condition is true ) ;
```

Do While Loop

1. This means that do-while would execute its statements at least once, even if the condition fails for the first time.
2. main( )

```
{
    do
    {
    printf ( "Hello World \n") ;
    } while ( 4 < 1 ) ;
}
```

3. If we use **while(4>1)** it becomes an infinite loop

Infinite Loop

1. An infinite loop is a looping construct that does not terminate the loop and executes the loop forever.
2. It is also called an indefinite loop or an endless loop.
3. It either produces a continuous output or no output.
4. It can be an error
5. Or by design
6. When to use?
   - when it is not known beforehand how many times the statements in the loop are to be executed
   - All the games. The game will accept the user requests until the user exits from the game.

Infinite Loop – By design

```
/* Execution of a loop an unknown number of times */
main( )
{
      char  another ;
      int  num ;
      do
      {
            printf ( "Enter a number " ) ;
            scanf ( "%d", &num ) ;
            printf ( "square of %d is %d", num, num * num ) ;
            printf ( "\nWant to enter another number y/n " ) ;
            scanf ( " %c", &another ) ;
      } while ( another == 'y' ) ;
}
```

Infinite Loop – By accident

```c
// Online C compiler to run C program online
#include <stdio.h>

int main() {
    int i=0;
    while(i<10){
        printf("Hello\n");
    }
    return 0;
}
```

Break statement

1. To jump out of a loop instantly, without waiting to get back to the conditional test.
2. We have used this in switch case
3. The keyword **break** allows us to do this.
4. When break is encountered inside any loop, control automatically passes to the first statement after the loop.
5. A break is usually associated with an if.

Without Break statement - Example

```c
// Online C compiler to run C program online
#include <stdio.h>

int main() {
    int i;
    for (i = 0; i < 10; i++) {
        if (i == 4) {
            // break;
        }
        printf("%d\n", i);
    }
    return 0;
}
```

```
/tmp/o4rFTZ4N78.o
0
1
2
3
4
5
6
7
8
9
```

Break statement - Example

```c
#include <stdio.h>
int main() {
    int i;
    for(i=0;i<=10;i++){
        if(i>5)
            break;
        printf("%d\n",i);
    }
    printf("Hello world");
    return 0;
}
```

```
/tmp/qtuWAEbMeU.o
0
1
2
3
4
5
Hello world
```

Break statement - Example

```c
// Online C compiler to run C program online
#include <stdio.h>

int main() {
    int i;
    for (i = 0; i < 10; i++) {
        if (i == 4) {
            break;
        }
        printf("%d\n", i);
    }
    return 0;
}
```

```
/tmp/o4rFTZ4N78.o
0
1
2
3
```

Break from accidental Infinite loop

```c
// Online C compiler to run C program online
#include <stdio.h>

int main() {
    int i=0;
    while(i<10){
        printf("Hello\n");
        break;
    }
    return 0;
}
```

```
/tmp/o4rFTZ4N78.o
Hello
```

Continue statement

1. In some programming situations we want to take the control to the beginning of the loop, bypassing the statements inside the loop, which have not yet been executed.
2. The keyword continue allows us to do this.
3. When continue is encountered inside any loop, control automatically passes to the beginning of the loop.

Continue statement – Example 1

```c
#include <stdio.h>
int main() {
    int i;
    for(i=0;i<=10;i++){
        if(i<5)
            continue;
        printf("%d\n",i);
    }
    printf("Hello world");
    return 0;
}
```

```
/tmp/qtuWAEbMeU.o
5
6
7
8
9
10
Hello world
```

Continue statement – Example 2

```c
#include <stdio.h>
int main() {
    int i;
    for(i=0;i<=10;i++){
        if(i<3)
            printf("%d\n",i);
        else if(i>7)
            printf("%d\n",i);
        else
            continue;
    }
    printf("Hello world");
    return 0;
}
```

```
/tmp/qtuWAEbMeU.o
0
1
2
8
9
10
Hello world
```

Continue statement – Example 3

```c
#include <stdio.h>
int main() {
    int i;
    for(i=0;i<10;i++){
        if(i%2==0){
            printf("%d\n",i);
        }
        else
            continue;
    }
    printf("Hello world");
    return 0;
}
```

```
/tmp/qtuWAEbMeU.o
0
2
4
6
8
Hello world
```

Break & Continue – Example

```c
#include <stdio.h>
int main() {
    int i, sum = 0;
    for(i=1;i<=10;i++){
        if(i%2==0)
            printf("%d\n",i);
        else{
            if(i==9)
                break;
            else
                continue;
        }
    }
    printf("Hello world");
return 0;
}
```

```
/tmp/4hhnDhreoy.o
2
4
6
8
Hello world
```

Nesting of Loops

1. If a loop exists inside the body of another loop, it's called a nested loop.
2. Any number of loops can be defined inside another loop, i.e., there is no restriction for defining any number of loops

Nesting of Loops - Example

```c
#include <stdio.h>
int main() {
    int i,j;
    for(i=0;i<=3;i++){
        for(j=0;j<=3;j++){
            printf("i = %d, j = %d\n",i,j);
        }
    }
    return 0;
}
```

```
/tmp/5r8rV551on.o
i = 0, j = 0
i = 0, j = 1
i = 0, j = 2
i = 0, j = 3
i = 1, j = 0
i = 1, j = 1
i = 1, j = 2
i = 1, j = 3
i = 2, j = 0
i = 2, j = 1
i = 2, j = 2
i = 2, j = 3
i = 3, j = 0
i = 3, j = 1
i = 3, j = 2
i = 3, j = 3
```

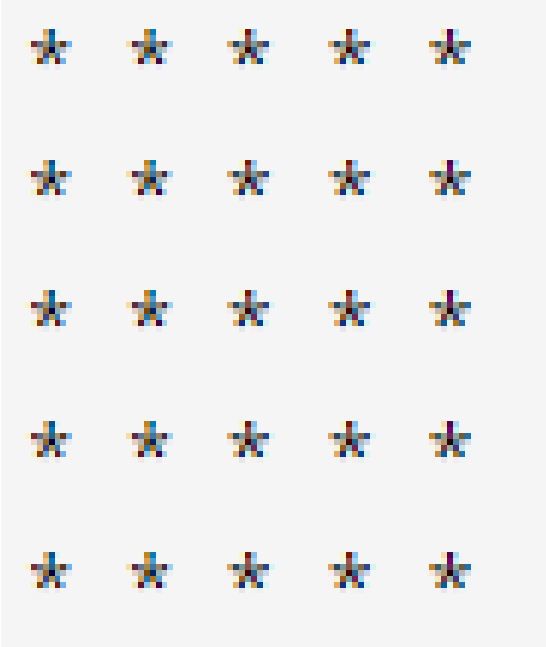
Nesting of Loops - Example

```c
#include <stdio.h>
int main() {
    int i,j;
    for(i=1;i<=3;i++){
        for(j=3;j>=1;j--){
            printf("i = %d, j = %d\n",i,j);
        }
    }
    return 0;
}
```

```
/tmp/5r8rV551on.o
i = 1, j = 3
i = 1, j = 2
i = 1, j = 1
i = 2, j = 3
i = 2, j = 2
i = 2, j = 1
i = 3, j = 3
i = 3, j = 2
i = 3, j = 1
```

Nesting of Loops - Example

1. Printing 2D figures on screen




```c
int main() {
    int i,j;
    for(i=0;i<5;i++){
        for(j=0;j<5;j++){
            printf("* ");
        }
        printf("\n");
    }

    return 0;
}
```

Nesting of Loops - Example

```
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
```

```c
int main() {
    int i,j;
    for(i=0;i<10;i++){
        for(j=0;j<i;j++){
            printf("* ");
        }
        printf("\n");
    }

    return 0;
}
```

What is the output? Explain

```c
#include <stdio.h>
int main() {
    int i=0,j=10;
    while(i<3){
        while(j>5){
            printf("i = %d, j = %d\n",i,j);
            j--;
        }
        i++;
        j=10;
    }
    return 0;
}
```

```
/tmp/5r8rV551on.o
i = 0, j = 10
i = 0, j = 9
i = 0, j = 8
i = 0, j = 7
i = 0, j = 6
i = 1, j = 10
i = 1, j = 9
i = 1, j = 8
i = 1, j = 7
i = 1, j = 6
i = 2, j = 10
i = 2, j = 9
i = 2, j = 8
i = 2, j = 7
i = 2, j = 6
```

What is the output? Explain

```c
#include <stdio.h>
int main() {
    int i=0,j=10;
    while(i<3){
        while(j>5){
            printf("i = %d, j = %d\n",i,j);
            j--;
        }
        i++;
    }
    return 0;
}
```

```
/tmp/5r8rV551on.o
i = 0, j = 10
i = 0, j = 9
i = 0, j = 8
i = 0, j = 7
i = 0, j = 6
```