

Lecture 6

Repetition

Today's Topics

- Loops
- Loop statements
- Jump statements
- Translating Flowchart to C Code

Loops

- The main idea of a loop is to **repeat an action or a series of actions**.

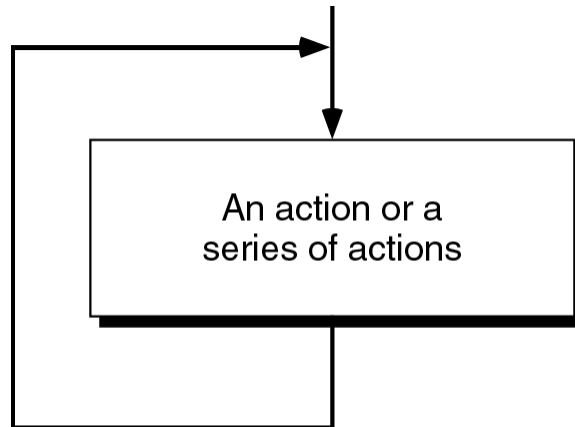
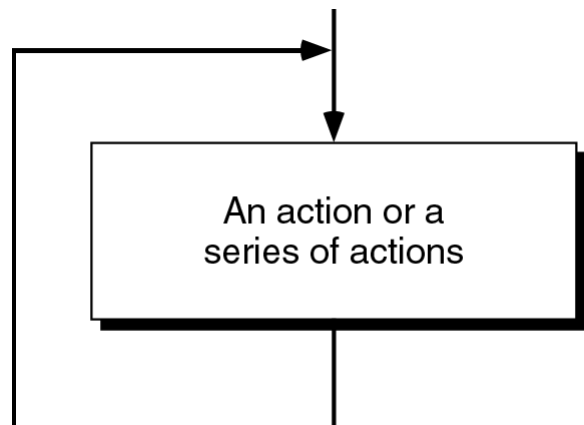


Figure 6-1: The concept of a loop

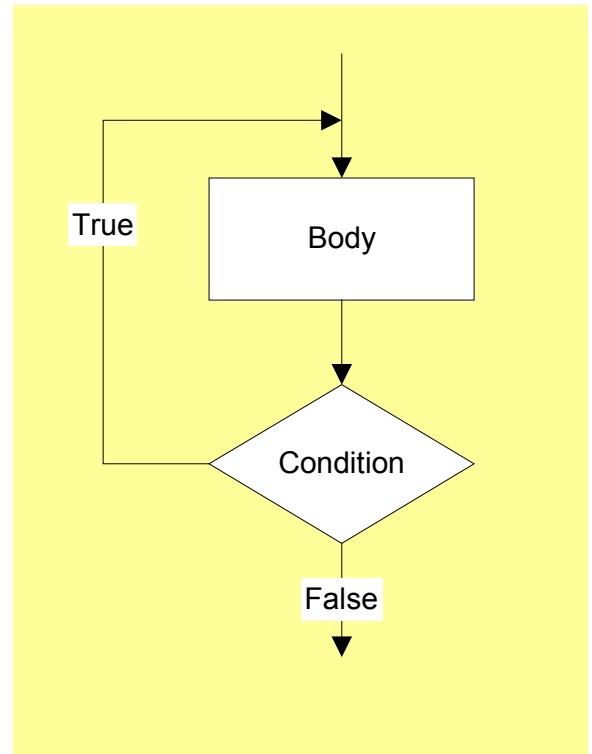
Loops

- But, when to stop looping?
- In the following flowchart, the action is executed over and over again. It never stop - This is called **infinite loop**
- Solution - put a **condition** to tell the loop either continue looping or stop.



Loops

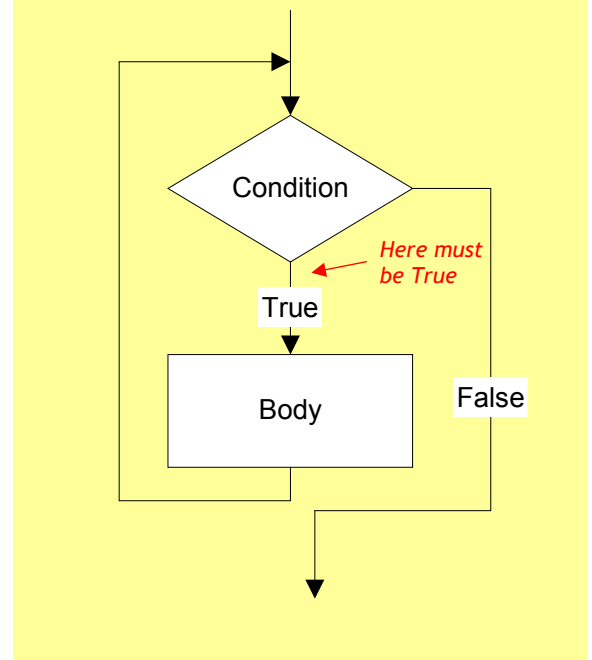
- A loop has two parts - **body** and **condition**
- **Body** - a statement or a block of statements that will be repeated.
- **Condition** - is used to control the iteration - either to continue or stop iterating.



Types of loop

- Two forms of loop - **pretest** loop and **post-test** loop.
- **Pretest loop**
 - the **condition is tested first**, before we start executing the body.
 - The body is executed if the condition is true.
 - After executing the body, the loop repeats

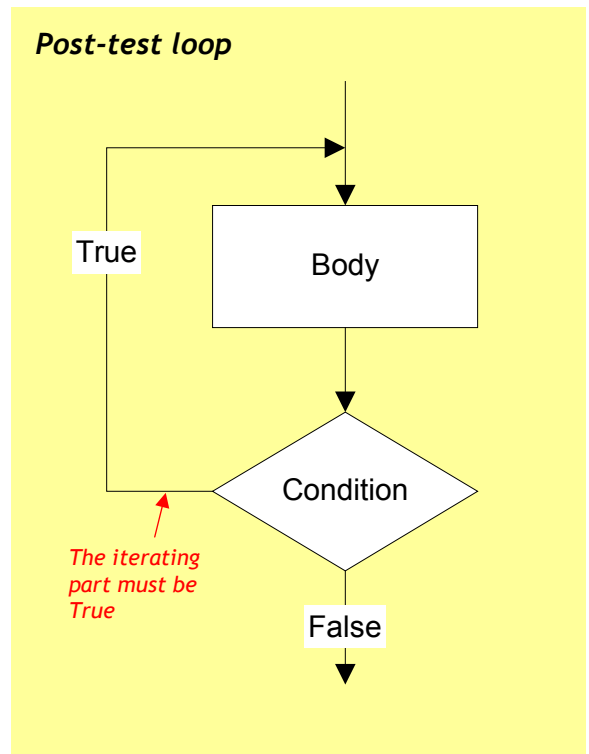
Pretest loop



Types of loop

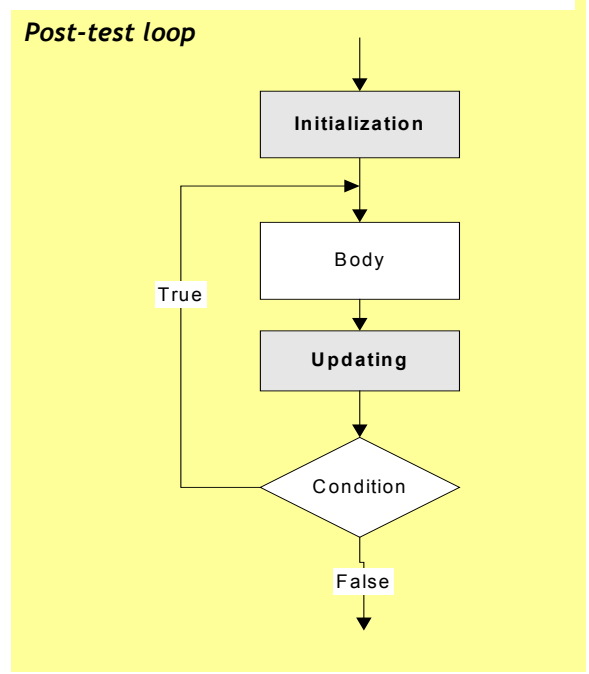
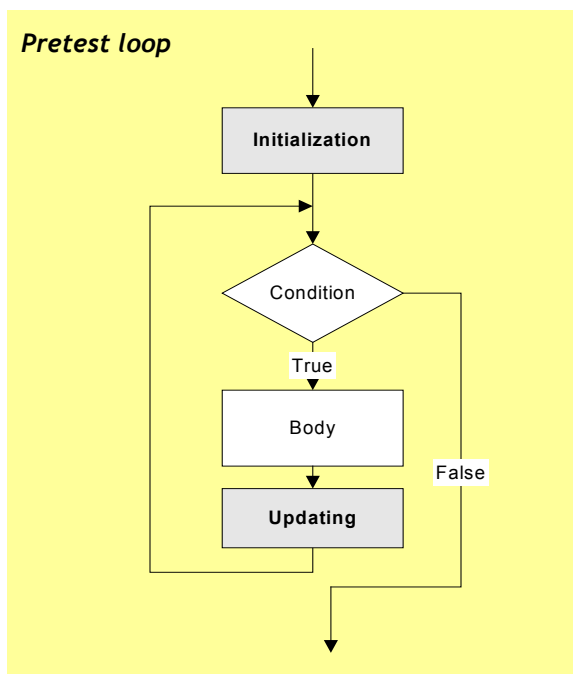
- **Post-test loop**

- the **condition is tested later**, after executing the body.
- If the condition is true, the loop repeats, otherwise it terminates.
- The body is always executed **at least once**.



Parts of a loop

- Beside the body and condition, a loop may have two other parts - **Initialization** and **Updating**



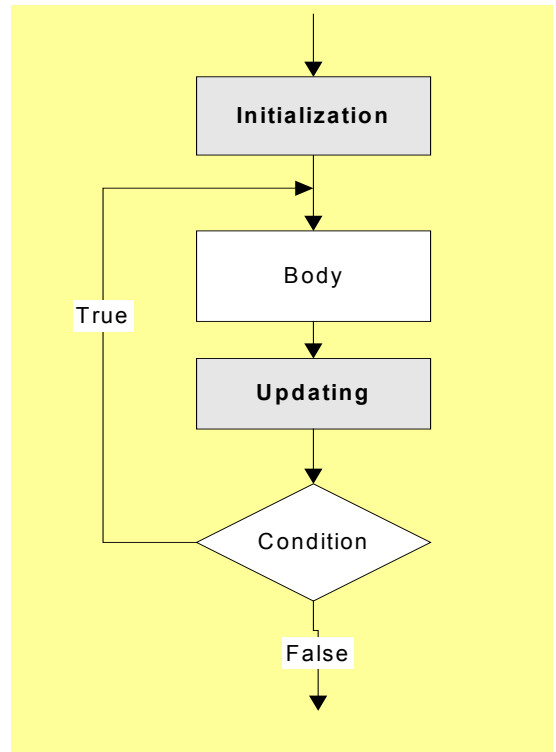
Parts of a loop

- **Initialization**

- is used to prepare a loop before it can start - usually, here we **initialize the condition**
- The initialization must be written outside of the loop - before the first execution of the body.

- **Updating**

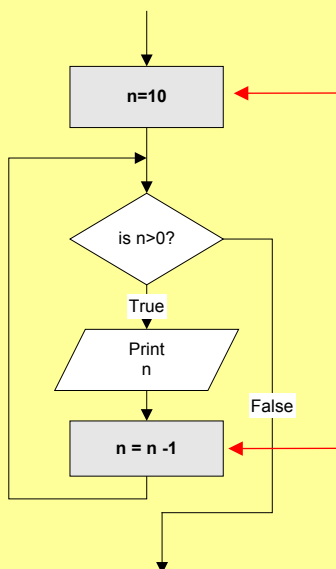
- is used to **update the condition**
- If the condition is not updated, it always true => the loop always repeats - an **infinite loop**
- The updating part is written inside the loop - it is actually a part of the body.



Parts of a loop

Example: These flowcharts print numbers 10 down to 1

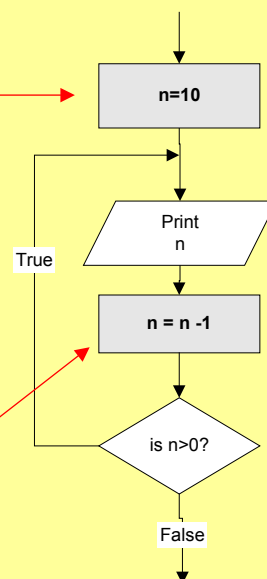
Pretest loop



Initialize n before start the loop

Every time the loop repeats, n is updated

Post-test loop



Loop statements

- C provides three loop statements:

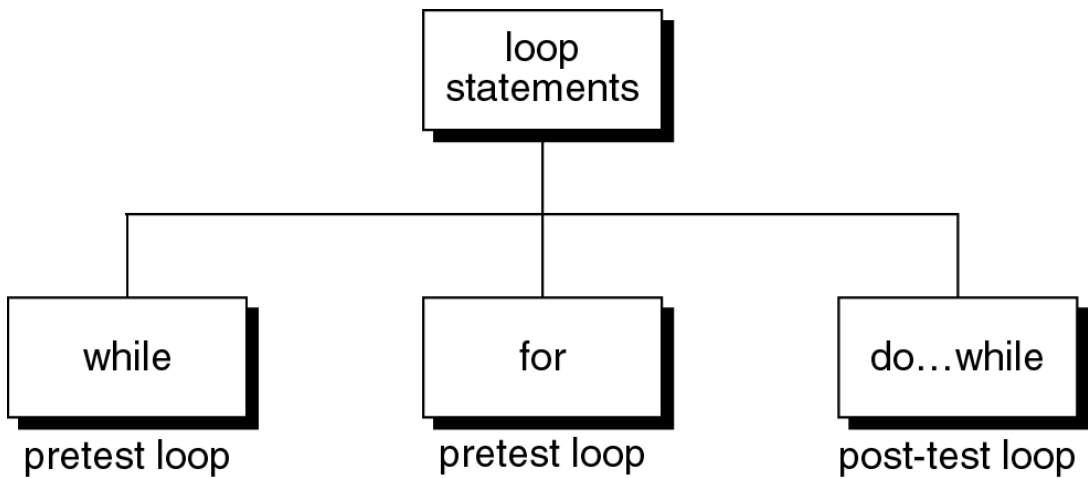
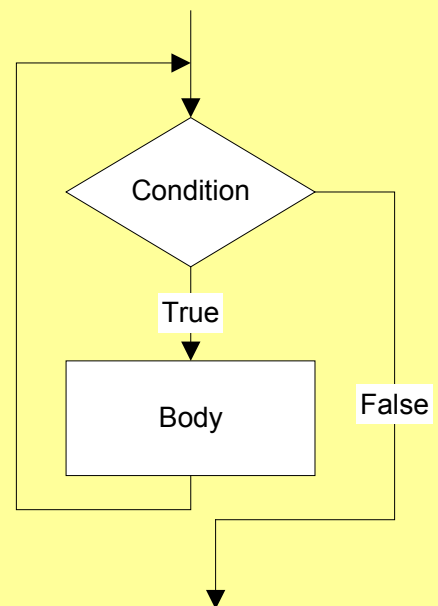


Figure 6-9: C loop constructs

while statement

```
while (Condition)  
{  
    Body;  
}
```

while flowchart



while statement

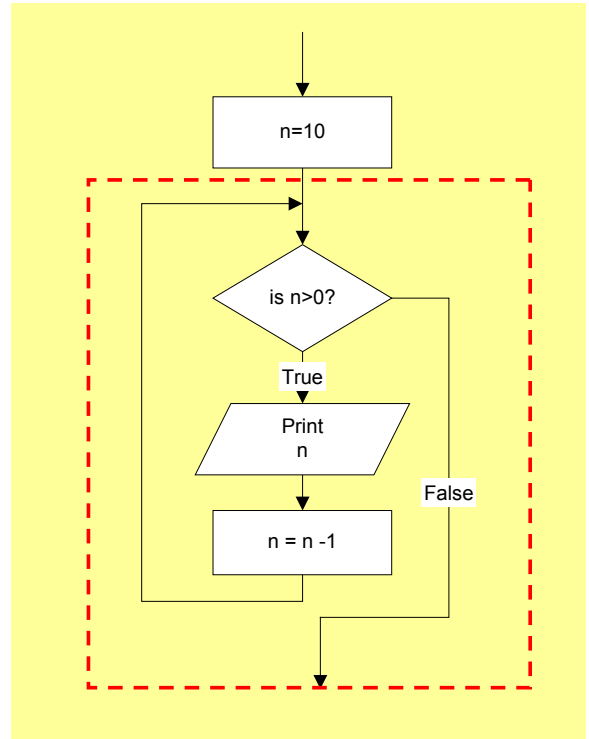
Example: This while statement prints numbers 10 down to 1

Note that, the first line (n=10) is actually not a part of the loop statement.

```
n=10;
while (n>0)
{
    printf("%d ",n);
    n=n-1;
}
```

Output:

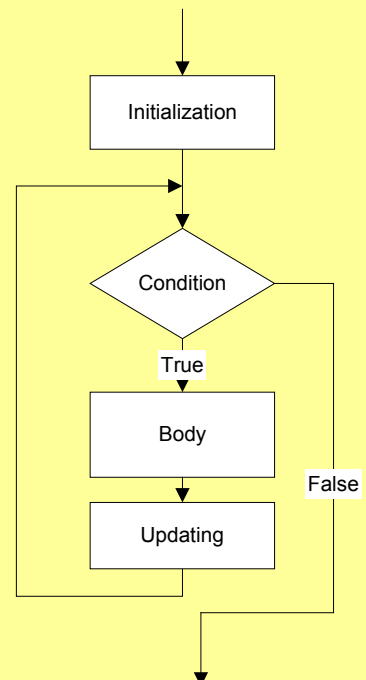
```
10 9 8 7 6 5 4 3 2 1
```



for statement

```
for (Initialization; Condition; Updating)
{
    Body;
}
```

for flowchart



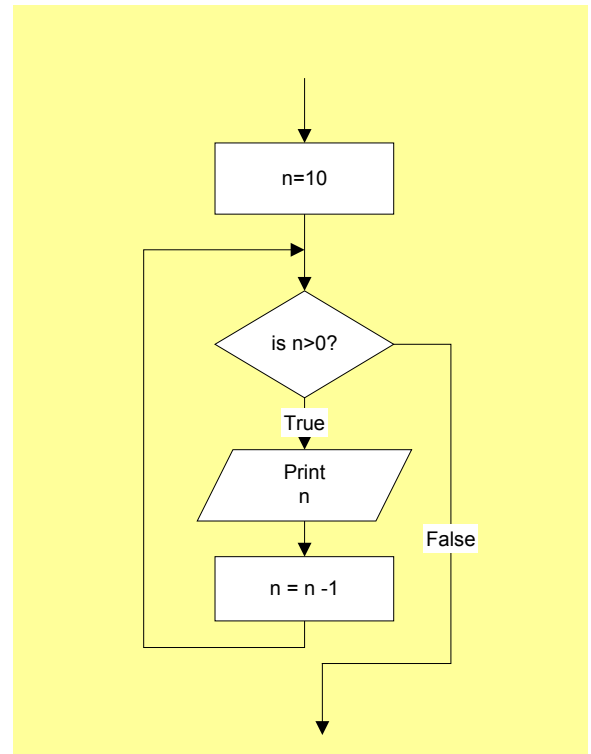
for statement

Example: This *for* statement prints numbers 10 down to 1

```
for (n=10; n>0; n=n-1)
{
    printf("%d ", n);
}
```

Output:

10 9 8 7 6 5 4 3 2 1



for vs. while statements

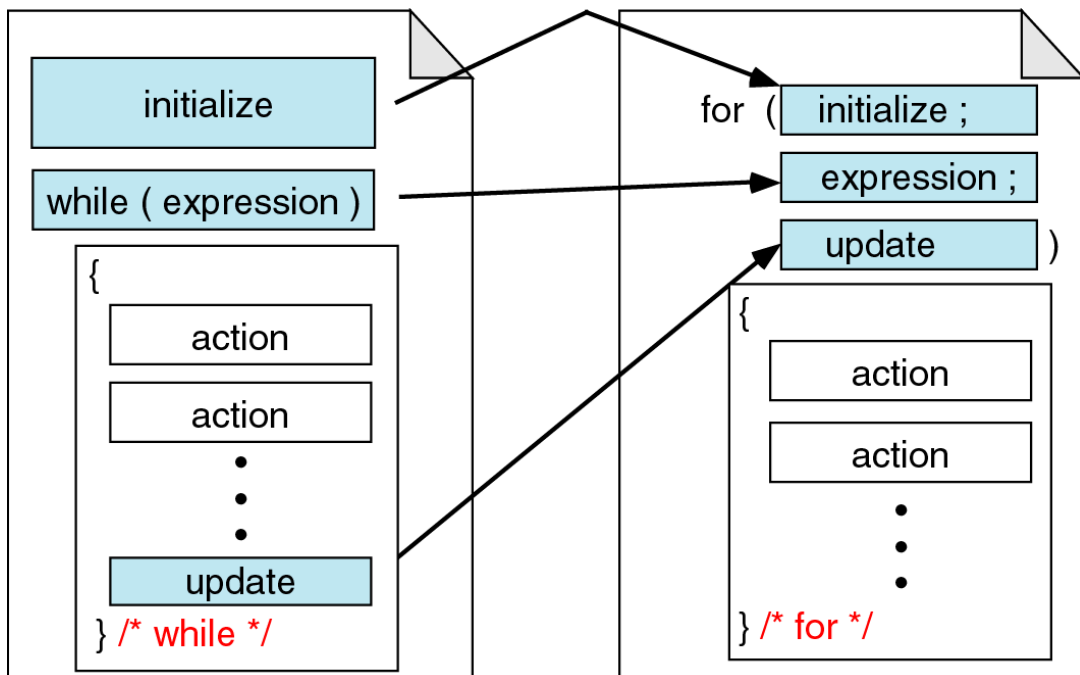
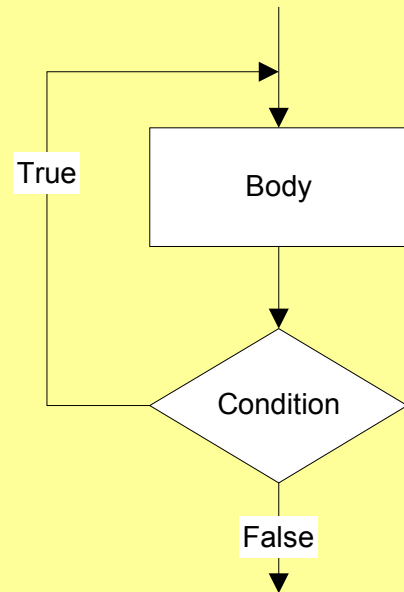


Figure 6-14: Comparing *for* and *while* loops

do...while statement

```
do
{
    Body;
} while (Condition);
```



do...while statement

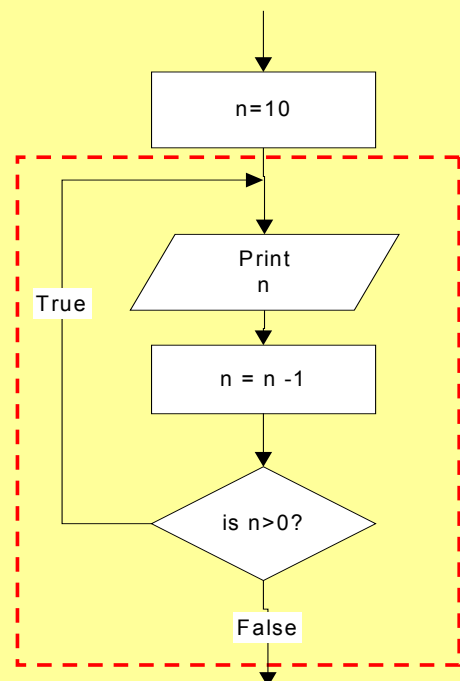
Example: This do...while statement prints numbers 10 down to 1

Note that, the first line (n=10) is actually not a part of the loop statement.

```
n=10;
do
{
    printf("%d ",n);
    n=n-1;
} while (n>0);
```

Output:

```
10 9 8 7 6 5 4 3 2 1
```



Loop statements

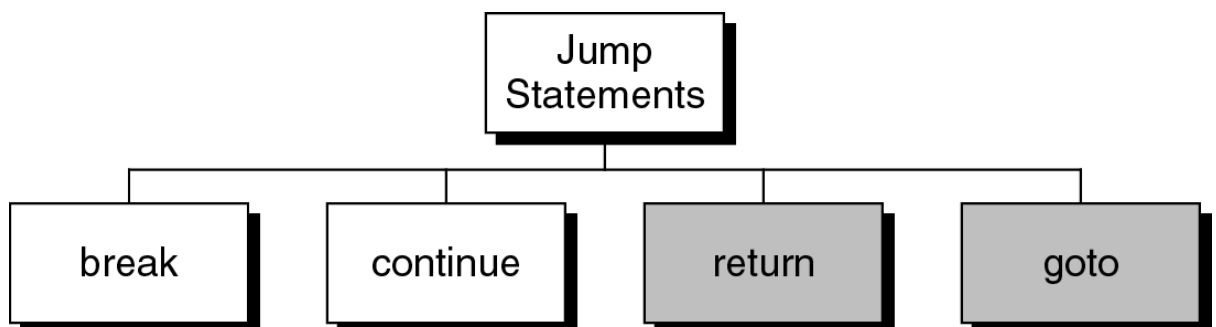
- If the body part has only **one statement**, then the bracket symbols, `{ }` may be omitted.
- Example: These two `for` statments are equivalent.

```
for (n=10; n>0; n=n-1)
{
    printf("%d ",n);
}
```

```
for (n=10; n>0; n=n-1)
    printf("%d",n);
```

Jump statements

- We know that, the repetition of a loop is controlled by the loop condition.
- C provides another way to control the loop, by using **jump statements**.
- There are four jump statements:



break statement

- It causes a loop to **terminate**

Example:

```
for (n=10; n>0; n=n-1)
{
    if (n<8) break;
    printf("%d ",n);
}
```

Output:

10 9 8

break statement

```
while (condition)
{
    ...
    for ( ...; ...; ... )
```

```
{
    ...
    if (otherCondition)
        break;
    ...
} /* for */
```

```
/* more while processing */
```

```
} /* while */
```

The break statement takes you out of the inner loop (the *for* loop). The *while* loop is still active.

Figure 6-21: *break* an inner loop

continue statement

- In `while` and `do...while` loops, the `continue` statement transfers the control to the loop condition.
- In `for` loop, the `continue` statement transfers the control to the updating part.

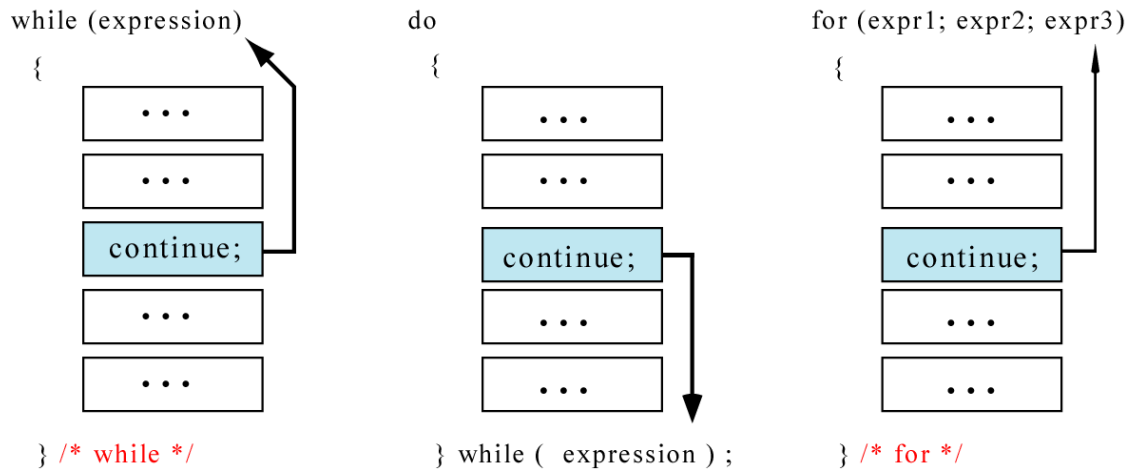


Figure 6-22: The *continue* statement

continue statement

Example:

```
for (n=10; n>0; n=n-1)
{
    if (n%2==1) continue;
    printf("%d ",n);
}
```

Output:

```
10 8 6 4 2
```

continue statement

Example:

```
n = 10;
while (n>0)
{
    printf("%d ",n);
    if (n%2==1) continue;
    n = n -1;
}
```

Output:

10 9 9 9 9 9

The loop then prints number 9 over and over again. It never stops.

return statement

- You have learned this statement in lecture 4 - Function.
- It causes a **function to terminate**.

Example:

```
void print_numbers()
{ int n=10;
  int i;

  while (n>0)
  {
    for (i=n;i>0; i--)
    {
      if (i%2==1) continue;

      if (i%4==0) break;

      if (n==6) return;

      printf("%d ", i);
    }
    printf("\n");
    n=n-1;
  }
}
```

The continue statement transfers control to the updating part (i--)

The break statement terminates the for loop.

The return statement terminates the function and returns to the caller.

Output:

10

6

return statement

- When to use `return`?
- *Example*: the following functions are equivalent

```
float calc_point(char grade)
{
    float result;

    if (grade=='A') result = 4.0;
    else if (grade=='B') result = 3.0;
    else if (grade=='C') result = 2.5;
    else if (grade=='D') result = 2.0;
    else result = 0.0;

    return result;
}
```

```
float calc_point(char grade)
{
    if (grade=='A') return 4.0;
    if (grade=='B') return 3.0;
    if (grade=='C') return 2.5;
    if (grade=='D') return 2.0;
    return 0.0;
}
```

The *else* part of each *if* statement may be omitted. It has never been reached.

return statement

```
float calc_point3(char grade)
{
    float result;

    switch (grade)
    {
        case 'A': result = 4.0;
                 break;

        case 'B': result = 3.0;
                 break;

        case 'C': result = 2.5;
                 break;

        case 'D': result = 2.0;
                 break;

        default: result = 0.0;
    }

    return result;
}
```

```
float calc_point4(char grade)
{
    switch (grade)
    {
        case 'A': return 4.0;

        case 'B': return 3.0;

        case 'C': return 2.5;

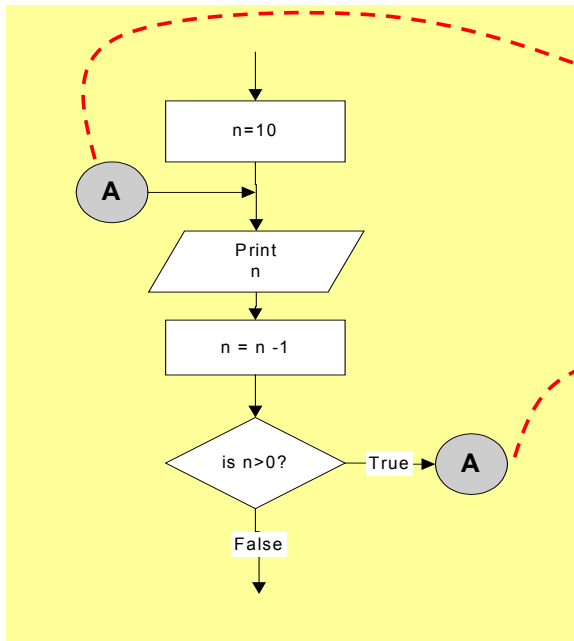
        case 'D': return 2.0;
    }
    return 0.0;
}
```

The *break* statement of each *case* may be omitted. It has never been reached.

goto statement

- It is used to translate **connector symbols** - jump to another part inside a program.
- But, it is not recommended to use - **it may cause unstructured programs**.

Example:



```
n=10;
A:
printf("%d ",n);
n = n -1;

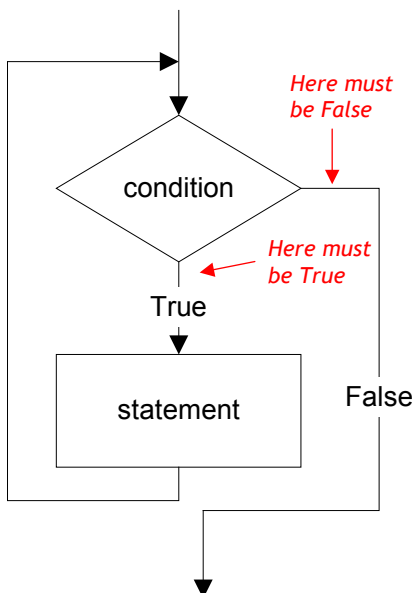
if (n>0) goto A;
```

Output:

```
10 9 8 7 6 5 4 3 2 1
```

Translating flowchart to C code

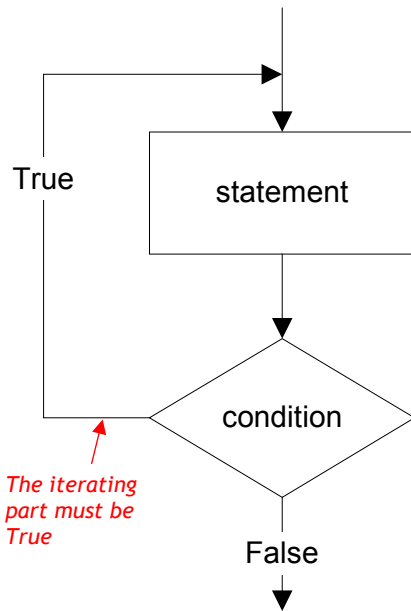
Pattern 1



```
while (condition)
{
    statement;
}
```

Translating flowchart to C code

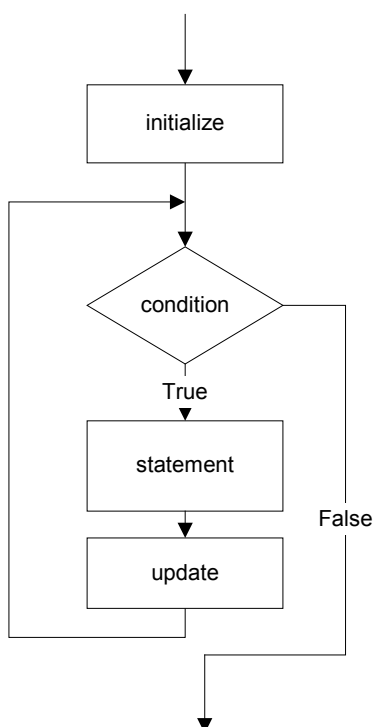
Pattern 2



```
do
{
    statement;
} while(condition);
```

Translating flowchart to C code

Pattern 3



```
for (initialize; condition; update)
{
    statement;
}
```

or

```
initialize;
while (condition)
{
    statement;
    update;
}
```