The switch Statement

- This causes a particular group of statements to be chosen from several available groups.
  - Uses “switch” statement and “case” labels.
  - Syntax of the “switch” statement:

```
switch (expression)  {
  case expression-1: { …… }
  case expression-2: { …… }

  case expression-m: { …… }
  default: { …… }
}
```

where “expression” evaluates to int or char

Example

```
switch (letter)
{
   case 'A':
       printf ("First letter \n");
       break;
   case 'Z':
       printf ("Last letter \n");
       break;
   default :
       printf ("Middle letter \n");
       break;
}
```

The break Statement

- Used to exit from a switch or terminate from a loop.
  - Already illustrated in the previous example.
- With respect to “switch”, the “break” statement causes a transfer of control out of the entire “switch” statement, to the first statement following the “switch” statement block.

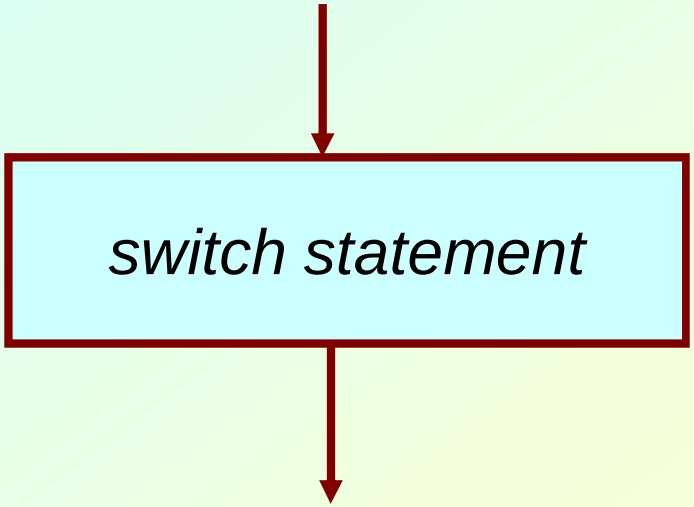
Example

```
switch  (choice = getchar())  {
   case 'r':
   case 'R': printf ("RED \n");
             break;
   case 'g':
   case 'G': printf ("GREEN \n");
             break;
   case 'b':
   case 'B': printf ("BLUE \n");
             break;
   default:  printf ("Invalid choice \n");
}
```

Example

```
switch  (choice = toupper(getchar()))  {

   case 'R': printf ("RED \n");
             break;
   case 'G': printf ("GREEN \n");
             break;
   case 'B': printf ("BLUE \n");
             break;
   default:  printf ("Invalid choice \n");

}
```

- **The "switch" statement also constitutes a single-entry / single-exit structure.**



A Look Back at Arithmetic Operators: the Increment and Decrement

Increment (++) and Decrement (--)

- **Both of these are unary operators; they operate on a single operand.**
- **The increment operator causes its operand to be increased by 1.**
  - **Example:** `a++, ++count`
- **The decrement operator causes its operand to be decreased by 1.**
  - **Example:** `i--, --distance`

- **Operator written before the operand (++i, --i))**
  - **Called pre-increment operator.**
  - **Operator will be altered in value *before* it is utilized for its intended purpose in the program.**
- **Operator written after the operand (i++, i--)**
  - **Called post-increment operator.**
  - **Operator will be altered in value *after* it is utilized for its intended purpose in the program.**

Examples

Initial values :: a = 10; b = 20;

`x = 50 + ++a;`	`a = 11, x = 61`
`x = 50 + a++;`	`x = 60, a = 11`
`x = a++ + --b;`	`b = 19, x = 29, a = 11`
`x = a++ – ++a;`	`Undefined value (implementation dependent)`

Called side effects:: while calculating some values, something else get changed.

Control Structures that Allow Repetition

Types of Repeated Execution

- **Loop**
  - Group of instructions that are executed repeatedly while some condition remains true.
- **Counter-controlled repetition**
  - Definite repetition – know how many times loop will execute.
  - Control variable used to count repetitions.
- **Sentinel-controlled repetition**
  - Indefinite repetition.
  - Used when number of repetitions not known.
  - Sentinel value indicates "*end of data*".

Counter-controlled Repetition

- **Counter-controlled repetition requires**
  - ***name*** **of a control variable (or loop counter).**
  - ***initial value*** **of the control variable.**
  - ***condition*** **that tests for the final value of the control variable (i.e., whether looping should continue).**
  - ***increment (or decrement)*** **by which the control variable is modified each time through the loop.**

Examples

```
int counter =1;                 /* initialization */

 while (counter <= 10) {     /* repetition condition */
       printf ("%d\n", counter );
       ++counter;                /* increment */
    }
```

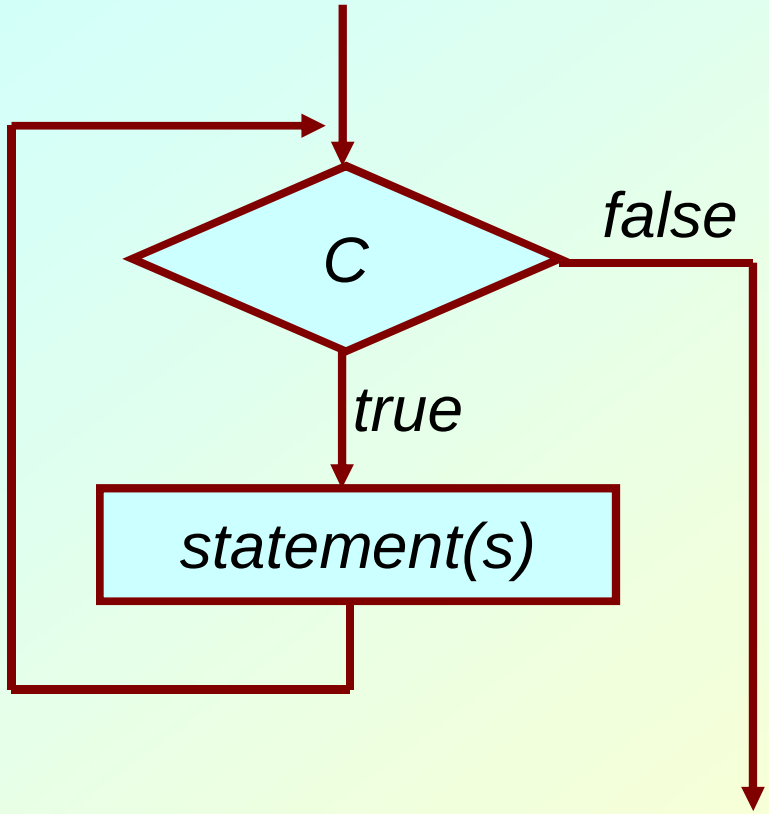
```
int counter;

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

while Statement

- The “while” statement is used to carry out looping operations, in which a group of statements is executed repeatedly, as long as some condition remains satisfied.

```
while (condition)
        statement_to_repeat;
```

```
while (condition)
{
        statement_1;
                ...
        statement_N;
}
```



Single-entry / single-exit structure

while :: Examples

```
int  digit = 0;

while  (digit <= 9)
  printf (“%d \n”, digit++);
```

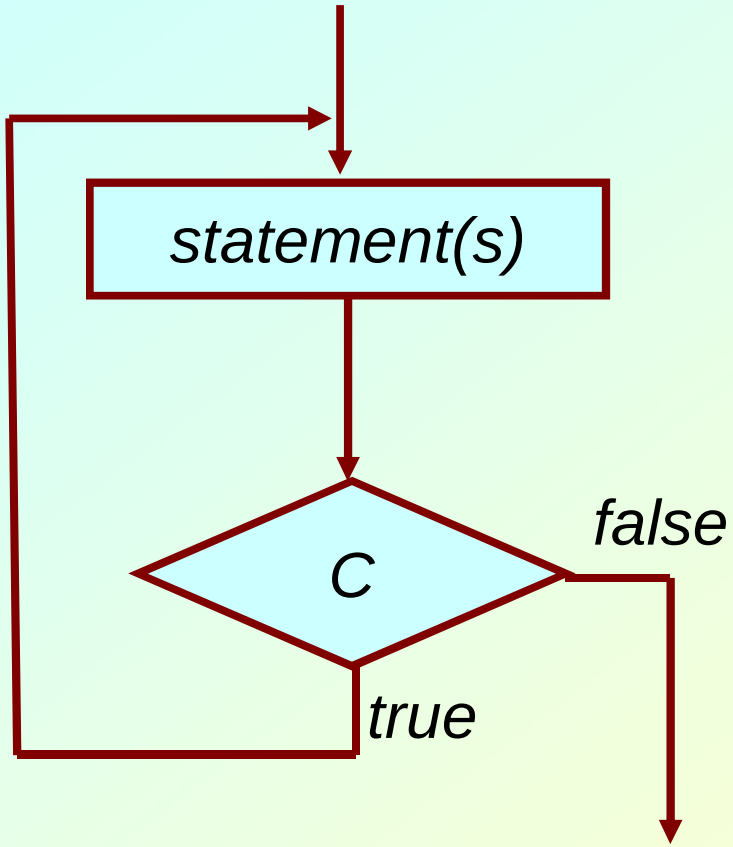
```
int  weight;

while ( weight > 65 )
{
   printf ("Go, exercise, ");
   printf ("then come back. \n");
   printf ("Enter your weight:");
   scanf ("%d", &weight);
}
```

do-while Statement

- **Similar to "while", with the difference that the check for continuation is made at the *end* of each pass.**
  - **In "while", the check is made at the *beginning*.**
- **Loop body is executed *at least once*.**

```
do
   statement_to_repeat;
while  (condition );
```

```
do {
        statement-1;
        statement-2;

        statement-n;
      }   while  (condition );
```



Single-entry / single-exit structure

do-while :: Examples

```
int  digit = 0;

do
   printf (“%d \n”, digit++);
while (digit <= 9);
```

```
int  weight;

do {
   printf ("Go, exercise, ");
   printf ("then come back. \n");
   printf ("Enter your weight: ");
   scanf ("%d", &weight);
}  while  (weight > 65 ) ;
```