```
main()
{…
 int  ans, val = 4;
```

Code

```
 val = val + 1;
 val++;
 ++val;
 ans = 2 * val++;
 ans = ++val / 2;
 val--;
 --val;
 ans = --val * 2;
 ans = val-- / 3;
 return 0;
}
```

val	ans
4	Garbage
5	
6	
7	
8	14
9	4
8	4
7	4
6	12
5	2

PRACTICE...

```
int a = 1, b = 2, c = 3, x;
x = ++a * b - c--;
```

// What is the value of x?

// What are the new values of a, b, and c?

MORE PRACTICE

What is the value of y?
What are the new values of a, b, c, and d?

```
int a = 1, b = 2, c = 3, d = 4, y;
y = ++b / c + a * d++;
```

```
y = 3/3 + 1*4
  = 1 + 4
  = 5
```

a	b	c	d
1	3	3	5

PRACTICE WITH ASSIGNMENT OPERATORS

```
int i = 1, j = 2, k = 3, m = 4;
```

Expression	Value
`i += j + k;`	
`j *= k = m + 5;`	
`k -= m /= j * 2;`	

PRACTICE WITH ASSIGNMENT OPERATORS

```
int i = 1, j = 2, k = 3, m = 4;
```

Expression	Value
`i += j + k;`	i=6
`j *= k = m + 5;`	k=9, j=18
`k -= m /= j * 2;`	m=1, k=2

COMPOUND STATEMENTS

- An expression such as *x = 0* or *j++* becomes a statement when it is followed by a semicolon ' ; '
- Braces { and } are used to group declarations and statements together into a *compound statement*.

```
{
    int j;
    j = 2 + 3;
    j++;
}   /*this entire thing now is a compound statement */
```

- A *Compound statement* is also called a *block*.

COMPOUND STATEMENT

- Variables can be declared in any block. *Discussion on this is deferred.*
- NB: There is no semicolon after the right brace that ends a block.

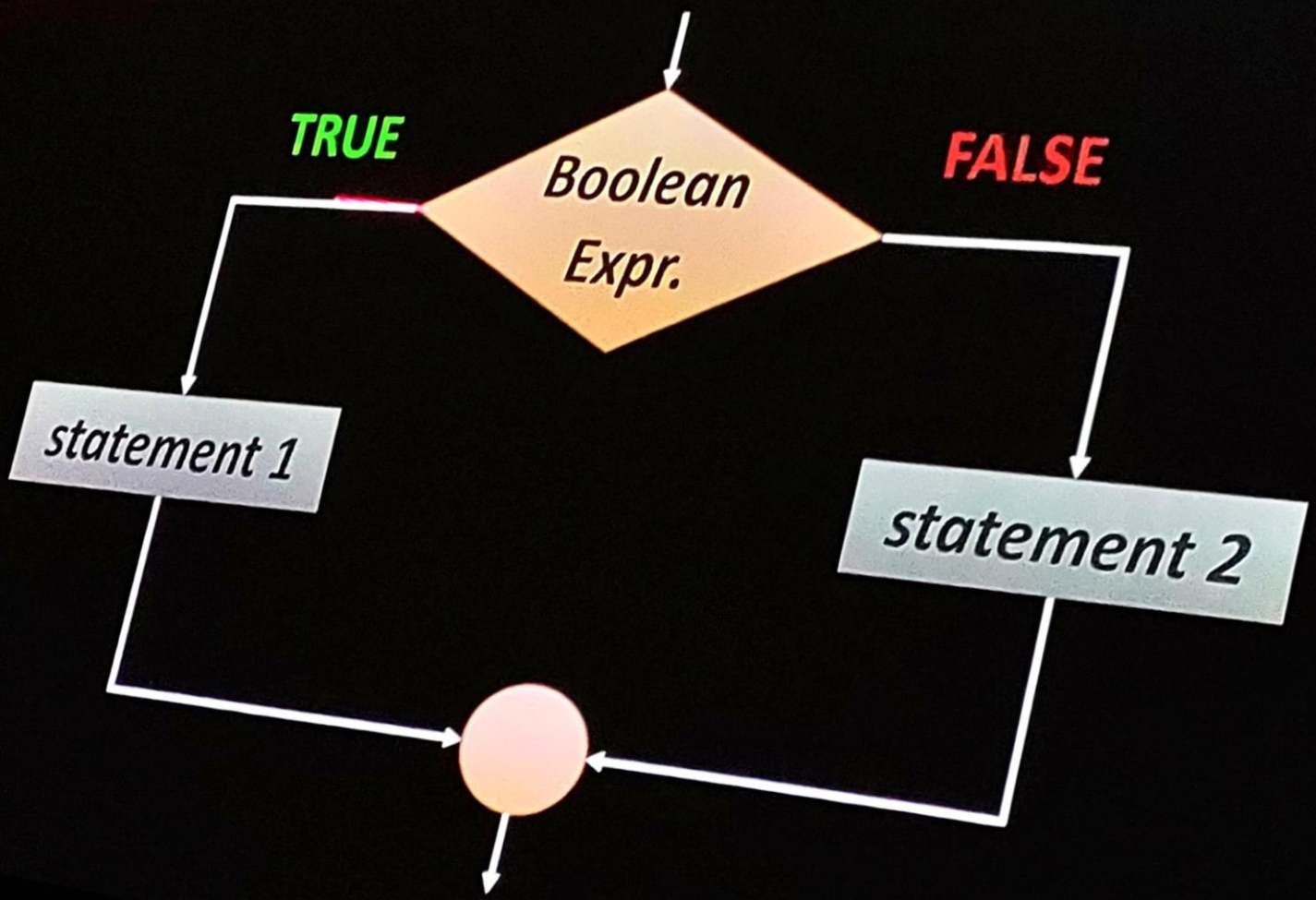
```
{
   int j;
   j = 2 + 3;
   j++;
}
```

```
if( condition )
{
    statement(s) /* body of the if statement */
}
```

The braces are not required if the body contains only a single statement.

However, C Coding Standards recommend their inclusion.

SELECTIVE EXECUTION: FLOW CHART



```
if( age >= 18 )
    {
        printf("You Vote!\n");
    }
```

Body: Good Practices: Always place braces around the body

```
if( value == 0 )
    {
        printf("You entered a zero, dumb-head!\n");
        printf ("Please try again.\n");
    }
```

Bo

IF-ELSE

- The *if-else* statement is used to express decisions.
- Formally the syntax is,
  - if(expression)
      statement1
    else
     statement2
  - The else part is optional
  - The expression is evaluated;
  - If it is TRUE (non-zero) then statement1 is executed, otherwise (if there is an else part) statement2 is executed.

IF-ELSE

- The *if-else* statement is used to express decisions.
- Formally the syntax is,
  - if(expression)
    statement1
    else
    statement2

- The else part is optional
- The expression is evaluated;
- If it is TRUE (non-zero) then statement1 is executed, otherwise (if there is an else part) statement2 is executed.
- *NB: if(x != 0)* is same as *if(x)*

EEK! SHARK ATTACK!

Shark infested waters

Someone is trapped in the jaws of a shark! Help!

No one warned us of a shark around!

Or did anyone?

BTW where is the Instructor?

Explain the error or else
I chomp your Instructor

Shark infested waters

IF-ELSE

WITHOUT BRACES

```
int  y = 5,  j,  k = 10;
if( y == 5 )
        k ++;
        j =  k * k;
else
        j = k;
```

ERROR

WITH BRACES

```
int  y = 5,  j,  k = 10;
if( y == 5 )
        {
                k++;
                j =  k * k;
        }
else
        j = k;
```

Explain the error or else chomp your Instructor

```
if(y == 5)
    k++;  /* The semicolon
ends the body of the if
statement altogether*/
```

```
if (y == 5)
{  k++;
   j = k *k;
}    /* The closing brace ends the if
part of the statement so an else
can follow.*/
```

WITHOUT BRACES

```
int  y = 5,  j,  k = 10;
if( y == 5 )
      k ++;
      j =  k * k;
else
      j = k;
```

ERROR


```
int  y = 5,  j,  k = 10;
if( y == 5 )
      {
            k++;
            j =  k * k;
      }
else
      j = k;
```

IF-ELSE PITFALLS

Check:

```
int j = 200;
if( j = 5)
  printf("A"); else
  printf("B");
```

- What is the output?

A

Because *j* has been assigned a value (=5) which is non-zero (TRUE).

NB: We are not checking whether *j* is equal to 5!

If that is so we should have used *if(j==5)*

This is a common pitfall. Beware!

IF-ELSE

```
if( n > 0 )
    if ( a > b)
        z = a;
else
    z = b;
```

```
if( n > 0 )
{
    if ( a > b)
        z = a;
    else
        z = b;
}
```

else associates with the closest previous else-less *if*.

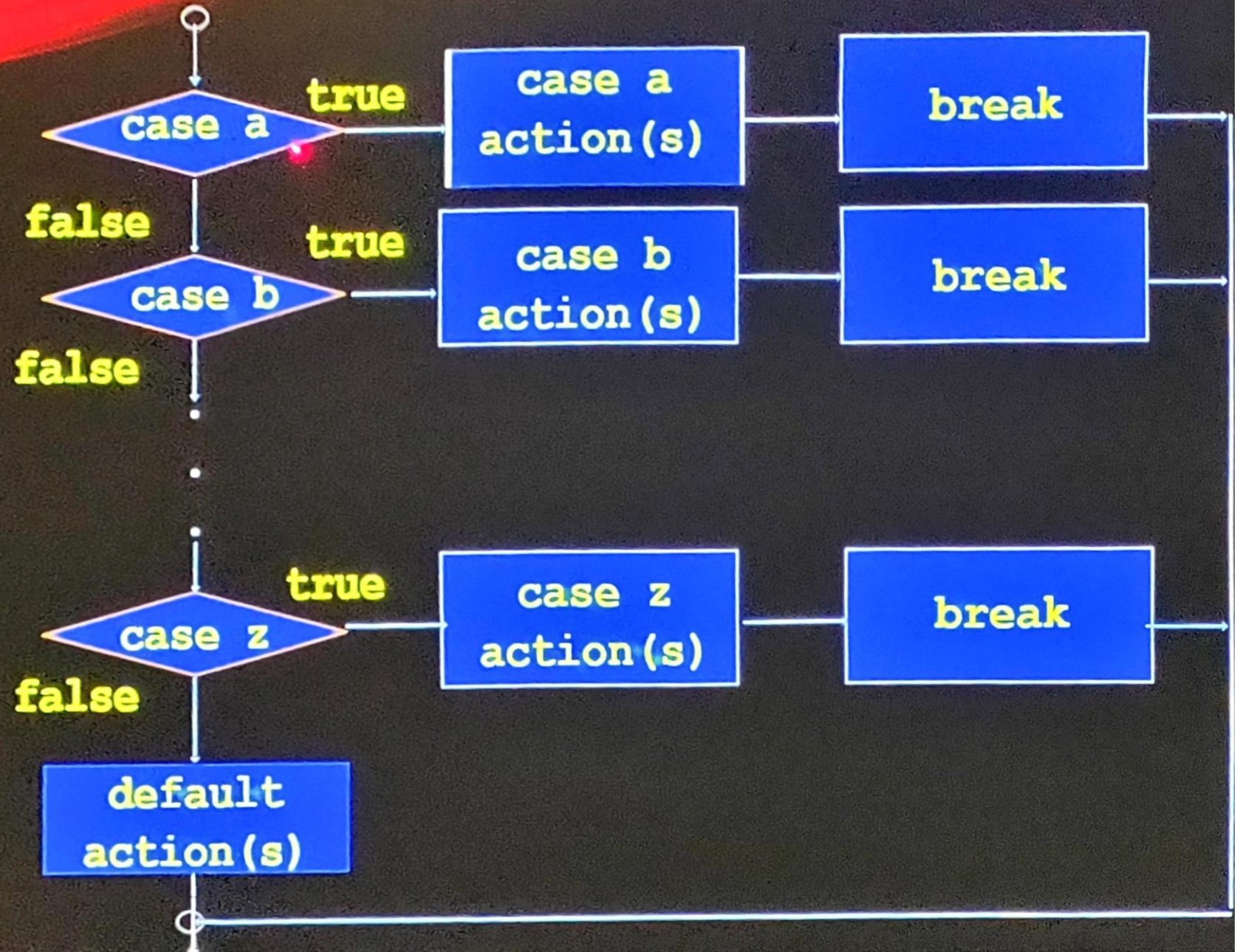
SWITCH

- The **switch** statement is a multi-way decision that tests whether an expression *matches one of a number of constant integer values* and branches accordingly.

```
switch(expression)
{    case const-expr_1 :  statements
     case const-expr_2 :  statements
     default: statements
}
```

The type of the expression should be either int or char.

FLOWCHART OF SWITCH STATEM



switch-case: A simple calculator

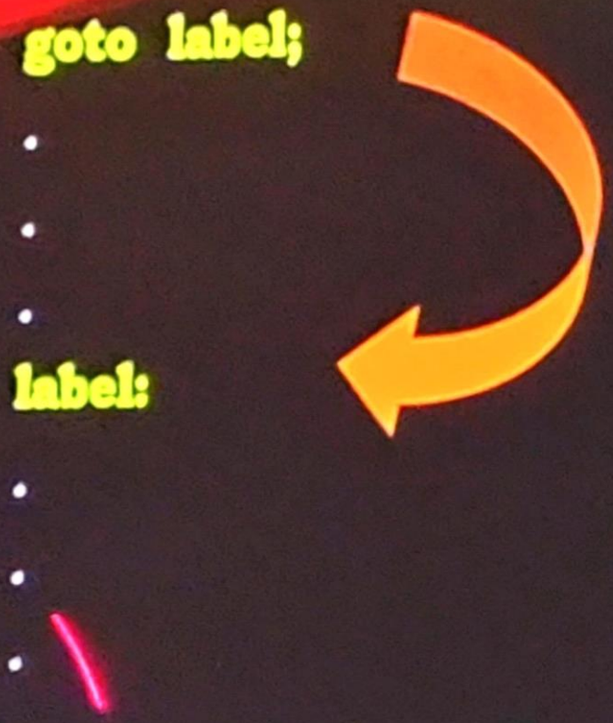

```
int main()
{
 float a=50.0, b=10.0, R;
 char choice;

 printf("Enter choice");
 scanf("%c",&choice);
 switch(choice)
 {
  case '+': R=a+b; printf("R=%f",R); break;
  case '-': R=a-b; printf("R=%f",R); break;
  case '*': R=a*b; printf("R=%f",R); break;
  case '/': R=a/b; printf("R=%f",R); break;
  default : printf("Wrong choice");
 }
 return 0;
}
```

switch-case: A simple calculator


```
switch(choice){
case '+' :
  //no break, works for both '+' & 'a'
  //next statement automatically get executed.
case 'a' : R=a+b; printf("R=%d",R); break;

case '-' :
case 's' : R=a-b; printf("R=%d",R); break;

case '*' :
case 'm' : R=a*b; printf("R=%d",R); break;

case '/' :
case 'd' : R=a/b; printf("R=%d",R); break;

default  : printf("Wrong choice");
}
```

```
int x;
scanf("%d",&x);
switch (x){
 case 1 ... 20:
    printf("You entered >=1 and <=20");
    break;
 case 21 ... 30:
     printf("You entered >=21 and <=30");
    break;
 default  :
    printf("You entered < 1 and >31") ;
    break;
 }
```

The goto Statement



/* label is similar to an identifier. This transfers control to that line unconditionally. */

goto: Example


```
get_input:
    printf( "Enter a and b: " );
    scanf( "%f %f", &a, &b );
    if( b == 0.0 )
    {   printf( "Trouble! Denominator is
        zero, enter values again\n" );
        goto get_input;
    }
    printf( "Result is … %f\n", a/b );
```

goto: Example...


```
get_value:
    printf( "Enter a and b: " );
    scanf( "%f,%f", &a, &b );
    if( b == 0.0 )
    {   printf( "Trouble! Denominator is
        zero, enter values again\n" );
        goto get_value;
    }
    printf( "Result is … %f\n",
```

10.0 3.0

Result is … ?????