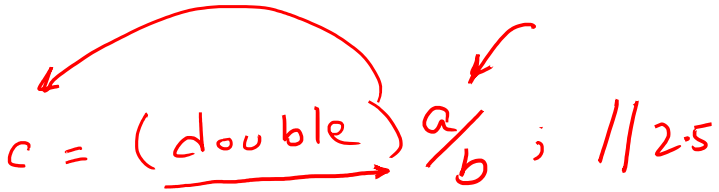
Fundamentals of Programming
CS-114

Lecture 4




```
int a ;
cin >> a ; // 2.5
cout << a ; // 2
int a = 5 , b = 2 ;
cout << a/b ; // 2
double c ;
c = a/b ; // 2.0
```

Arithmetic Expressions

- Arithmetic operations can be used to express the mathematic expression in C++:

$b^2 - 4ac$	$b*b-4*a*c$
$x(y+z)$	$x*(y+z)$
$\frac{1}{x^2+x+3}$	$1/(x*x+x+3)$
$\frac{a+b}{c-d}$	$(a+b)/(c+d)$

Polynomials in C++

- In algebra

$$m = \frac{a + b + c + d + e}{5}$$

- In c++

$$m = (a + b + c + d + e)/5$$

- What happens if I don't put the parenthesis ()?

Evaluate

In what order will the expression be evaluated?

Algebra $z = pr\%q + w/x - y$

C++ $z = p * r\%q + w/x - y$

Evaluate

In what order will the expression be evaluated?

Algebra: $z = pr\%q + w/x - y$

C++:

```
z = p * r % q + w / x - y;
```

6 1 2 4 3 5

Run Code

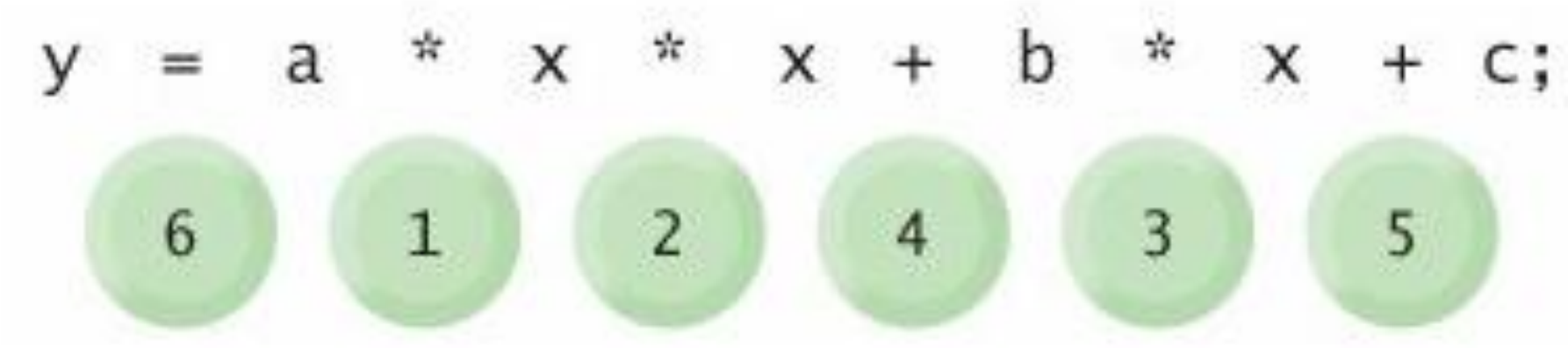
2nd Degree Polynomial

$$y = ax^2 + bx + c$$

```
y = a * x * x + b * x + c;
```

2^{nd} Degree Polynomial

$$y = ax^2 + bx + c$$



Run Code

Arithmetic Assignment Operators

Type	Operators	Usage
Arithmetic assignment	'+=' '-=' '*=' '/=' '%='	a+=b is the same as a=a+b a-=b a*=b a/=b a%=b

```
a = a + b;
a += b;
a += b;
```

Relational and Equality Operators

Standard algebraic equality or relational operator	C++ equality or relational operator	Sample C++ condition	Meaning of C++ condition
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y

x == y
↓
bool

(Deitel and Deitel (5th Ed), fig 2.12)

Logical Operators

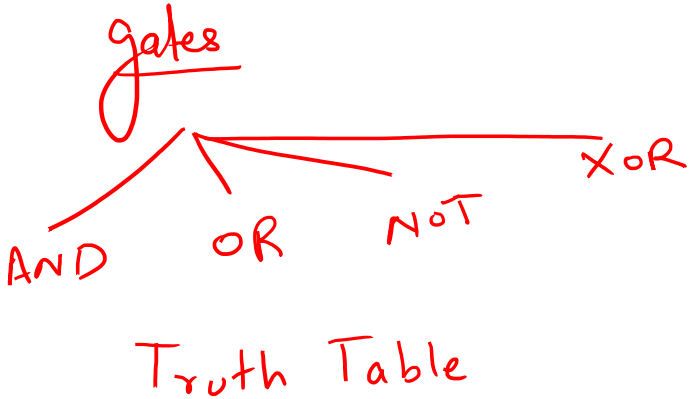
Type	Operators	Usage
Logical	'&&' '||'	

AND OR

- Logical operators are carried out on statements, e.g. statement1 && statement 2, etc.
- Logical AND (&&)
  - false && false= false
  - false && true = false
  - true && false= false
  - true && true = true
- Logical OR (||)

conditional statement

AND



Logical Operators

- Logical OR (||)
  - false || false = false
  - false || true = true
  - true || false = true
  - true || true = true
- Logical NOT (!)
  - !false = true
  - !true = false

Expression	True or False
(6 <= 6) && (5 < 3)	False
(6 <= 6) || (5 < 3)	True
(5 != 6)	T
(5 < 3) && (6 <= 6) || (5 != 6)	F
(5 < 3) && ((6 <= 6) || (5 != 6))	F
!((5 < 3) && ((6 <= 6) || (5 != 6)))	

!(F && (T || F))

!(F && T)

!(F) → T

Expression	True or False
(6 <= 6) && (5 < 3)	False
(6 <= 6) || (5 < 3)	True
(5 != 6)	
(5 < 3) && (6 <= 6) || (5 != 6)	
(5 < 3) && ((6 <= 6) || (5 != 6))	
!((5 < 3) && ((6 <= 6) || (5 != 6)))	

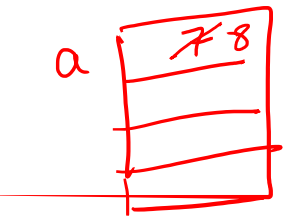
Expression	True or False
(6 <= 6) && (5 < 3)	False
(6 <= 6) || (5 < 3)	True
(5 != 6)	True
(5 < 3) && (6 <= 6) || (5 != 6)	
(5 < 3) && ((6 <= 6) || (5 != 6))	
!((5 < 3) && ((6 <= 6) || (5 != 6)))	

Expression	True or False
(6 <= 6) && (5 < 3)	False
(6 <= 6) || (5 < 3)	True
(5 != 6)	True
(5 < 3) && (6 <= 6) || (5 != 6)	True
(5 < 3) && ((6 <= 6) || (5 != 6))	
!((5 < 3) && ((6 <= 6) || (5 != 6)))	

Expression	True or False
(6 <= 6) && (5 < 3)	False
(6 <= 6) || (5 < 3)	True
(5 != 6)	True
(5 < 3) && (6 <= 6) || (5 != 6)	True
(5 < 3) && ((6 <= 6) || (5 != 6))	False
!((5 < 3) && ((6 <= 6) || (5 != 6)))	

Expression	True or False
(6 <= 6) && (5 < 3)	False
(6 <= 6) || (5 < 3)	True
(5 != 6)	True
(5 < 3) && (6 <= 6) || (5 != 6)	True
(5 < 3) && ((6 <= 6) || (5 != 6))	False
!((5 < 3) && ((6 <= 6) || (5 != 6)))	True

Unary Increment and Decrement Operators



a++;

++a;

Type	Operators	Usage
Increment and decrement	'++' '- -'	a++ is the same as a=a+1 a-- is the same as a=a-1

- Prefix
  - ++a;
  - --a;
- Postfix
  - a++;
  - a--;

a++; ← Postfix

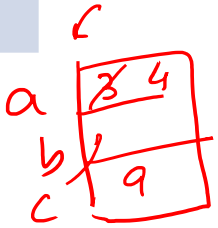
a--;

Prefix → ++a;

--a;

a++;

a=a+1;

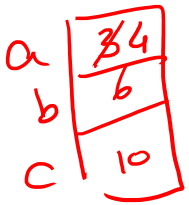


int a=3, b=6; c;

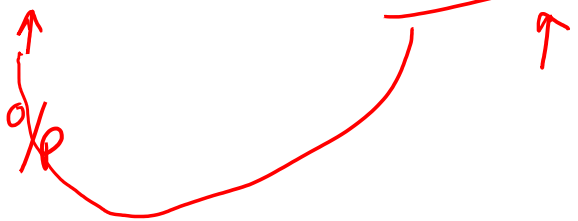
→ c = (a++) + b; // 9

// 3 + 6 → 9

c = ++a + b; // 10

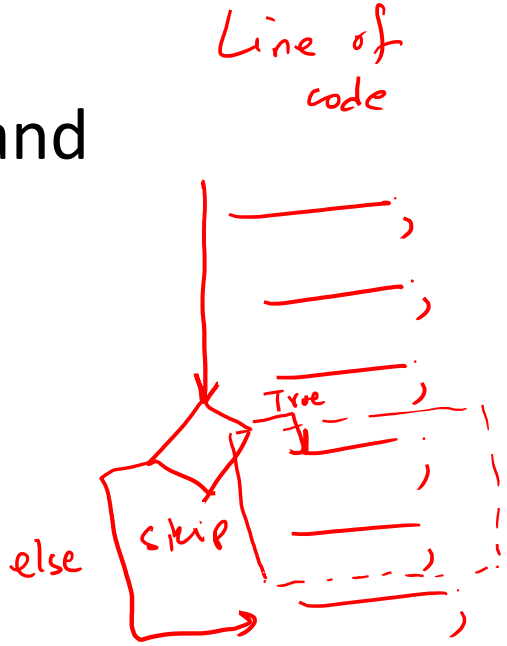


Selection/Conditional statements



Program types

- All programs can be written in terms of only three control structures, namely,
  - the sequence structure,
  - the selection structure (Branching) and
  - the repetition structure (Looping).

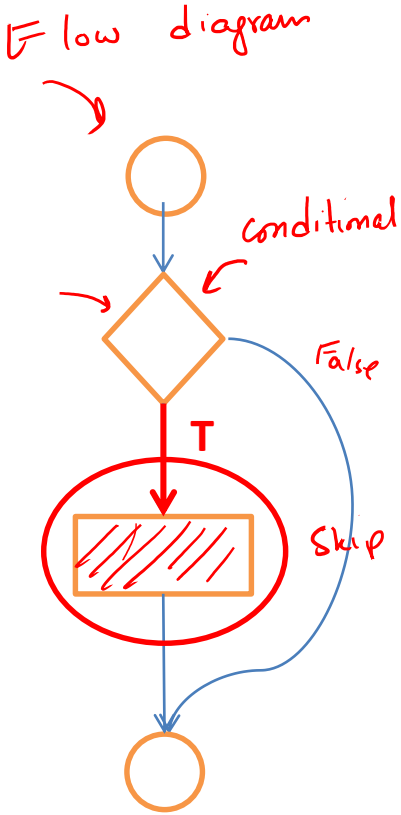


Selection/Conditional statements

- *Selection statements* cause one section of code to be executed or not depending on a *conditional clause*.
  - **if** statement
  - **switch** statement

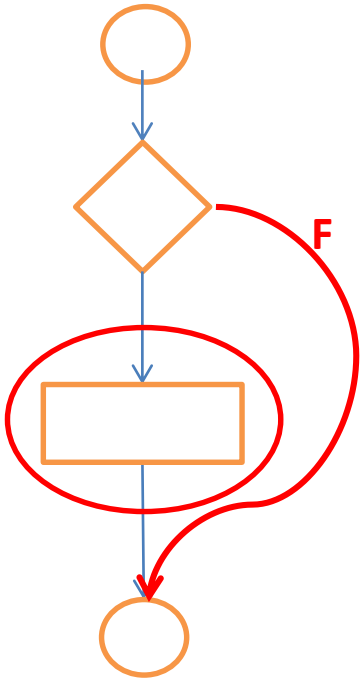
The **if** statement

- Used where the execution of an action depends on the satisfaction of a condition.
  - if condition is true -> action is done



The **if** statement

- Used where the execution of an action depends on the satisfaction of a condition.
  - if condition is true -> action is done
  - if condition is false -> action is not done



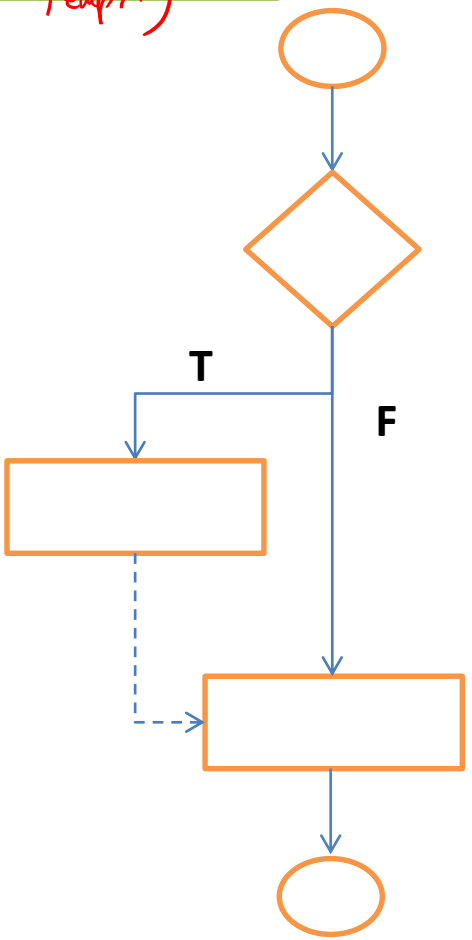
The **if** statement

Syntax

Logical or Relational

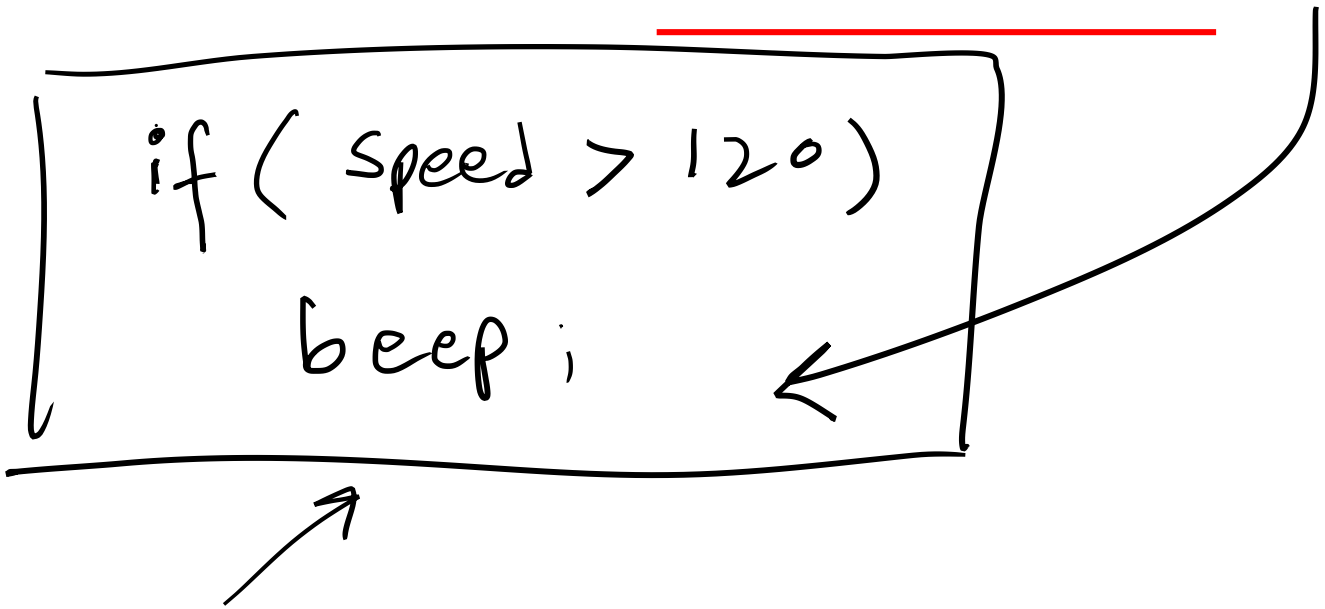
Any statement that results in **True** or **False**

```
if (condition) ;x
   { statement1;
     statement 2;
       ….
     statement n;
   }
```



The **if** statement

- Develop a system that tells the user that he is over-speeding if his speed crosses120km/hr.



The **if** statement

- Develop a system that tells the user that he is over-speeding if his speed crosses120km/hr.

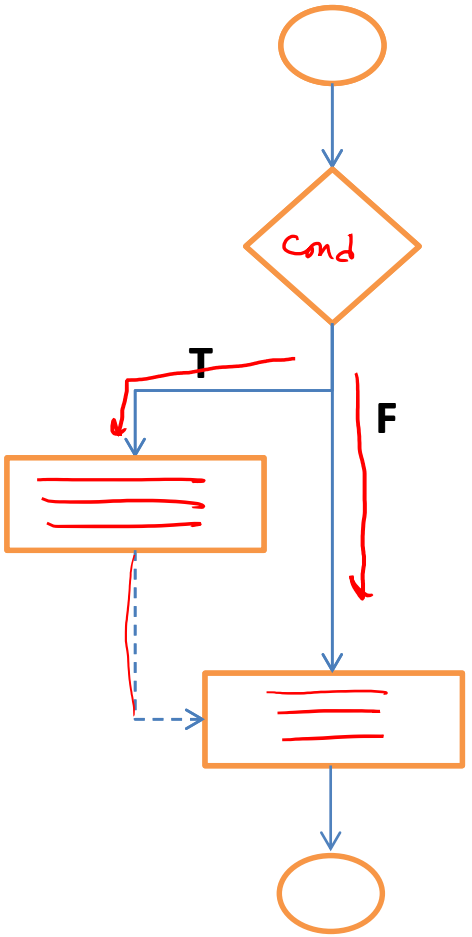
- Pseudocode

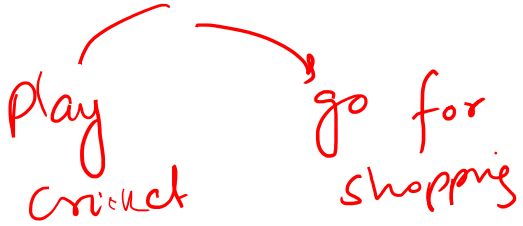
  IF speed is greater than 120km/hr

  THEN print “over-speeding”

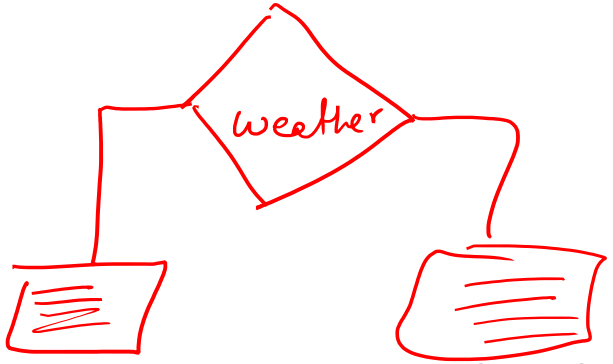
The **if** statement

```
if (speed>120)
     {cout<<"over speeding"<<endl;
       cout << "reduce speed now";
     }
…
…
…
```

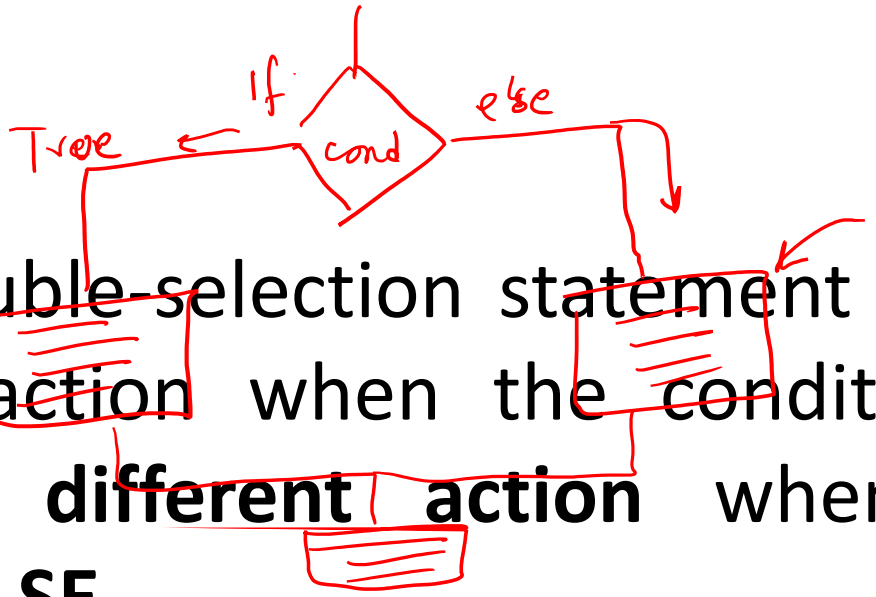




If - else



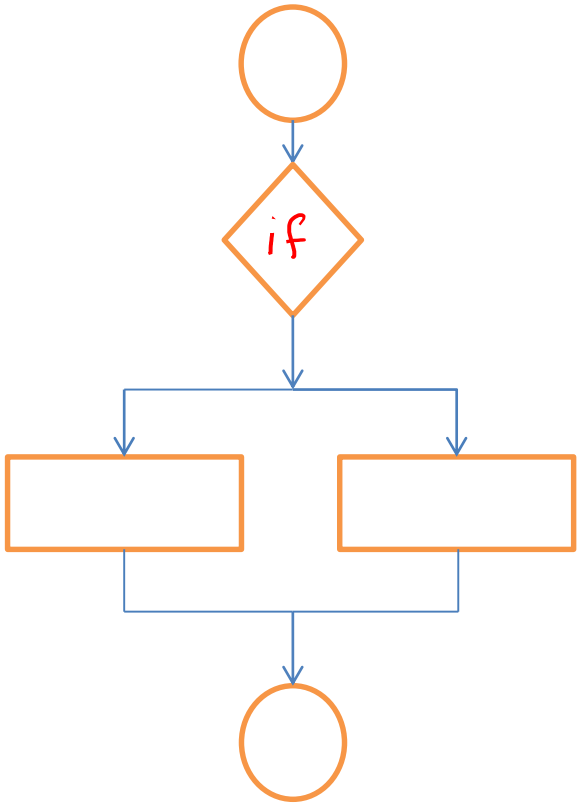
- **if** is a single-selection statement which performs an indicated action only when the condition is **TRUE**; otherwise the action is **skipped**.



- **if-else** is a double-selection statement which performs an action when the condition is **TRUE** and a **different action** when the condition is **FALSE**.

If - else

- Used where there are different actions to be executed depending on the result of a given condition.
  - if condition is true -> action is done
  - if condition is false -> a different action is done



If - else

- **Syntax:**

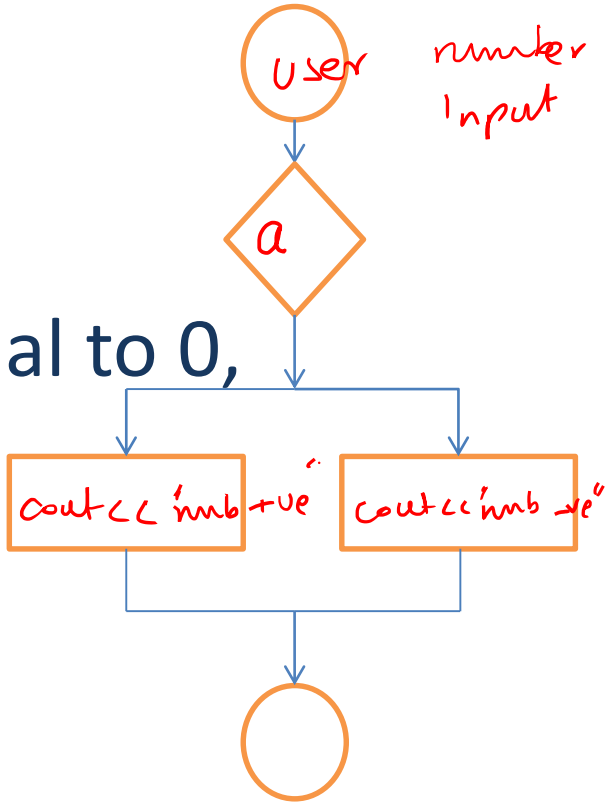
```
if (condition)
   {statement1; statement 2;…. statement n;
   }
else
   {statement1; statement 2;….statement n;
   }
```

- **Note:** if the specified condition is true, then all the statements in the first curly braces will be executed, else all the statements in the next curly braces will be executed.

If - else

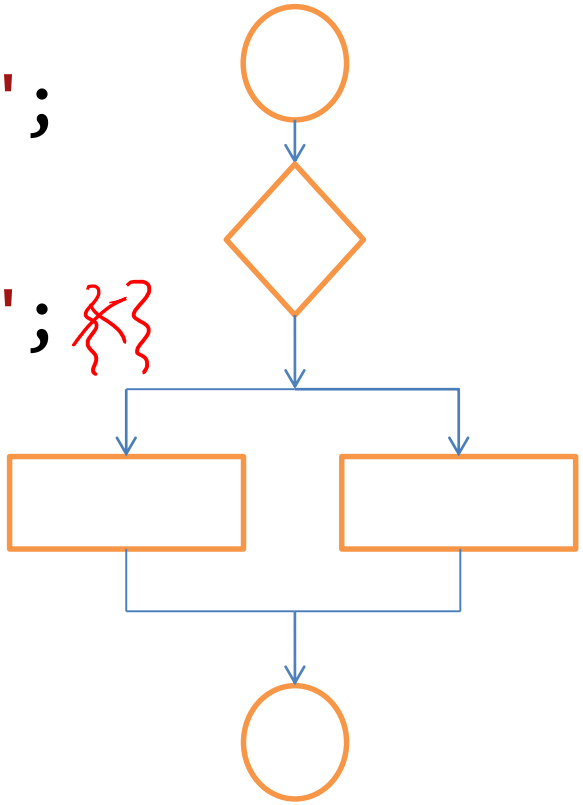
- Determine whether a number is positive or negative.

- Pseudocode
- IF number is greater than or equal to 0,
  THEN print "positive"
- ELSE print "negative"



If - else

```
if (number>=0)
    cout<<"number is positive";
else
    cout<<"number is negative";
```



NOTE: {} are not used here

If - else

- **TRY ME! :** for even numbers add half the number to y. For odd numbers, subtract half.

```
if (x%2==0)
    y += x/2;
else
    y -= x/2;
```

Some notes

- If there is only one statement following the ***if***, then the curly braces are optional
- If there are multiple statements, the braces are necessary
- Same is true for the statements following the ***else***

Some notes

– E.g.

```
if (a<b) //false
    a=b+1; //skip
    b = 10;
    cout<<"print A"<<a;  // print A 5
```

Output (if condition is true) false

- ***a = b+1;***
- ***the cout statement is ignored in this case because of the missing { }***

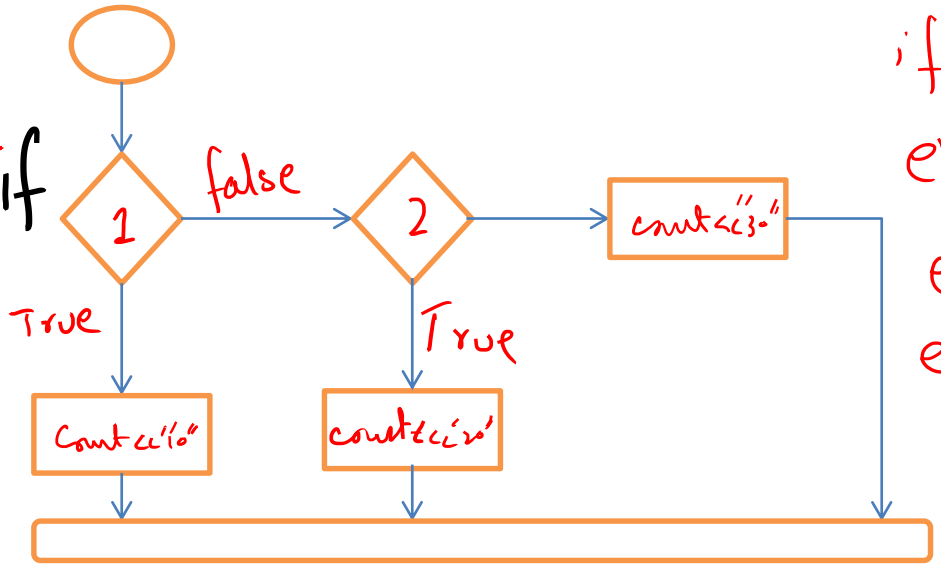
Some notes

- An *if* can exist without an *else*.
- An *else*, without an *if* has no meaning.

The **else if** keyword

- Used where there are multiple actions to be executed based on different conditions.
  - if condition is true -> action is done
  - if condition is false -> next condition is checked

```
if ( x > 0 )
    cout << "positive";
else if (x < 0)
    cout << "-ve";
else
    cout << "0";
```




```
if ( )
else if ( )
else if ( )
else if ( )
```

The **else if** keyword

- **Syntax:**

```
if (condition)
    {statement1; statement 2;…. statement n;
    }
else if (condition 2)
    {statement1; statement 2;….statement n;
    }
else
    {statement1; statement 2;….statement n;
    }
```

The **else if** keyword

```
if (x==0)
    {cout << " x is ZERO" <<endl;}
else if (x>0)
    {cout << " x is positive"
<<endl;}
else
    {cout << " x is negative"
<<endl;}
```

The **else if** keyword

- Additional alternative control paths
- Conditions evaluated in order until one is met; inner statement (or statements) are then executed
- If multiple conditions are true, **only first** condition is executed, the remaining are ignored

- if number is odd → even
  if " " even → odd

```
if ( x % 2 == 1 ) // odd
    x = x++; //even
else
    x = x++; //odd
```

if x number +ve & less than 10

cout → A

else

B

$0 < x \leq 10$

```
if( x<=10 && x>0 )
    cout << "A";
else
    cout << "B";
```

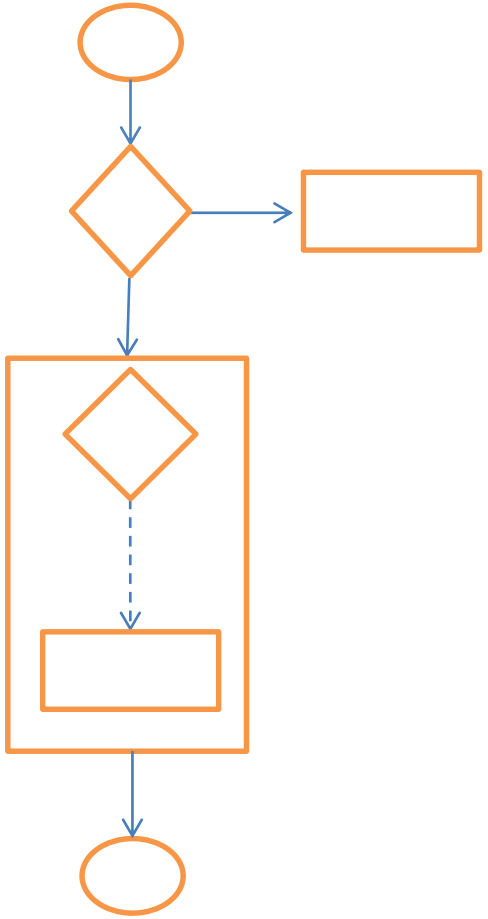
$x \leq 10$ ✗

$x > 0$ && $x \leq 10$ ✓

```
#include <iostream>
using namespace std;
int main()
{
    int a;
    cin >> a;
    if( a>0 && a<=10)
      cout << "A"; //strings
    else
      cout << "B";
    return 0;
}
```

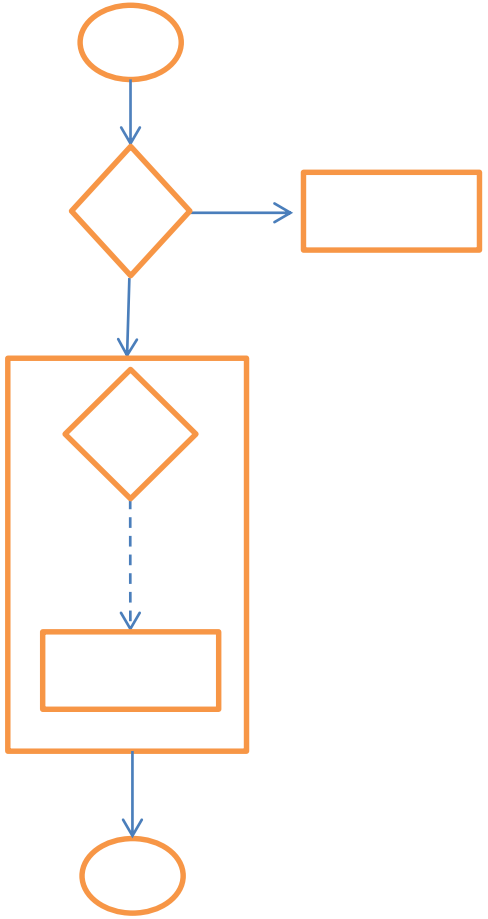
Nested **if** statements

- An if statement within the executable block of another if statement
- Use?
- Used where several conditions need to be met for the execution of an action.
- Is this completely true? How?



Nested **if** statements

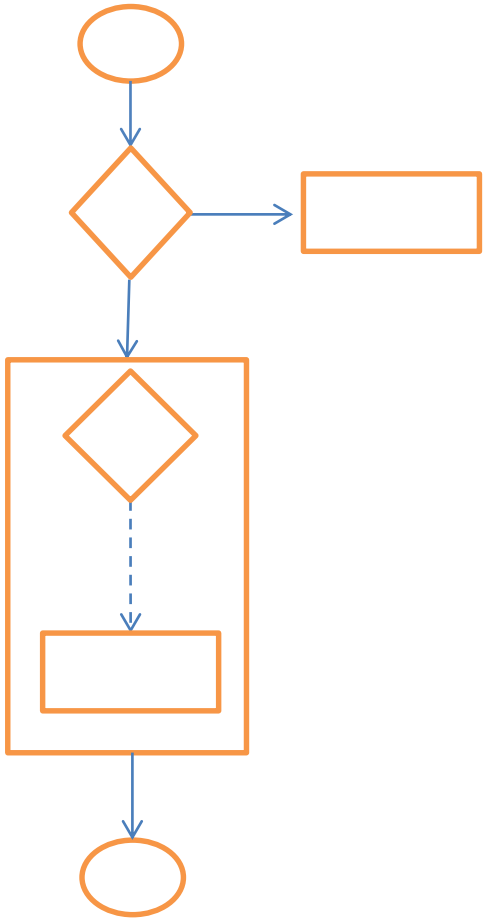
- Conditions are checked in order, i.e. the inner condition is checked only if the outer condition is satisfied.
  - if condition is true -> inner condition is checked
  - if inner condition is true -> action is done



Nested **if** statements

- **Syntax:**

```
if (condition)
    {if (condition)
          statement1;//(…. to n)
      else
          statement;
    }
else
    {statement1;statement 2;….
      statement n;
   }
```



Nested **if** statements

- Determine whether the number is positive even, positive odd, or zero.
- Pseudocode

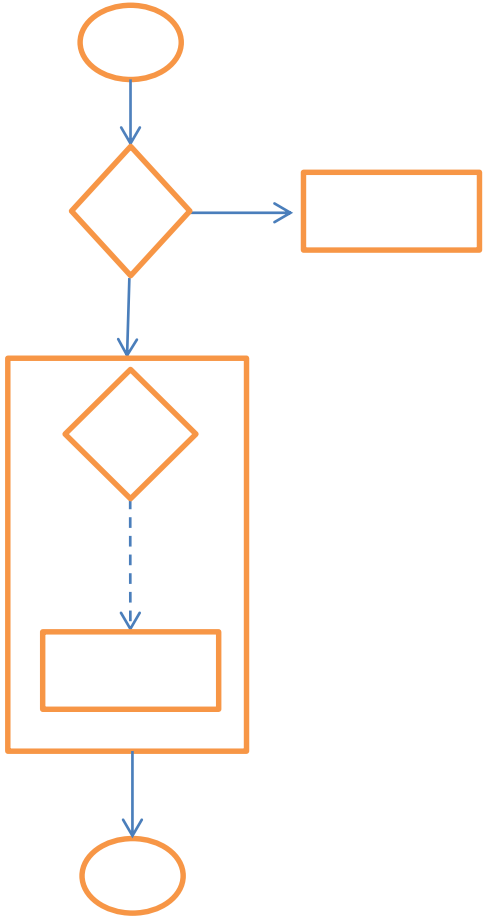
IF number is greater than 0,

    IF number is divisible by 2,

        THEN print “positive even”

    ELSE print “positive odd”

ELSE print “zero”



Nested **if** statements

```
if (num>0)
     {if ((num%2)==0)
          cout << "positive even" <<
          endl;
     else
          cout << "positive odd" <<
          endl;
     }
else
     cout << "zero" << endl;
```

Nested **if** statements

1.

```
if (x%4 == 0)
      {if (x%2 == 0)
        y= 2;
      }
else
    y= 1;
```

To associate else with outer if statement: use braces

2.

```
if (x < 0)
   if (y != 4)
      z = y * x;
   else
      z = y / x;
else if (y > 4)
   z = y + x;
else
   z = y - x;
```

If else-if

```
if (day == 1)
cout<< "Sunday”;

if (day == 2)
cout << "Monday”;

if (day == 3)
cout << "Tuesday”;

if (day == 4)
cout << “Wednesday”;
```

```
if (day == 1)
   cout<< "Sunday”;

else if (day == 2)
   cout<< "Monday”;

else if (day == 3)
   cout<< "Tuesday”;

else if (day == 4)
   cout <<"Wednesday”;
```

Here, each if condition will be checked even if the true one is found earlier on.

Here, the compiler will skip the remaining if conditions following the true one.

switch-statement Syntax

- switch (selector) // controlling expression

```
{
    case label_1:                    statement_Sequence_1
                                     break;
     case label_2:
                                     Statement_Sequence_2
                                     break;

     case label_n:
                                     Statement_Sequence_n
                                     break;
     default:
                                     Default_Statement_Sequence
}
```

The Controlling Statement

- A switch statement's controlling statement must return one of these types
  - A bool value
  - An enum constant
  - An integer type
  - A character
- The value returned is compared to the constant values after each "case"
  - When a match is found, the code for that case is used

The break Statement

- The break statement ends the switch-statement
  - Omitting the break statement will cause the code for the next case to be executed!
  - Omitting a break statement allows the use of multiple case labels for a section of code
    - case 'A':
      case 'a':
      cout << "Excellent.";
      break;
    - Runs the same code for either 'A' or 'a'

The default Statement

- If no case label has a constant that matches the controlling expression, the statements following the default label are executed
  - If there is no default label, nothing happens when the switch statement is executed
  - It is a good idea to include a default section

The **switch** statement

```
cout <<"Enter a number between 1 & 3";
cin >> a;
switch(a)
{
        case 1:
                cout <<"it is case 1\n";
                break;
        case 2:
                cout <<"it is case 2\n";
                break;
        case 3:
                cout <<"it is case 3\n";
                break;
        default:
                cout <<"it is default";
                break;
}
```

Home task (Try to do these without Loops)

- Program to Check input character as Vowel or Consonant
- Program to Find the Largest Number Among Three integers
- Program to Make a Simple Calculator to Add, Subtract, Multiply or Divide Using switch...case
- Write a program which takes 5 integers as input and sort them in ascending order.