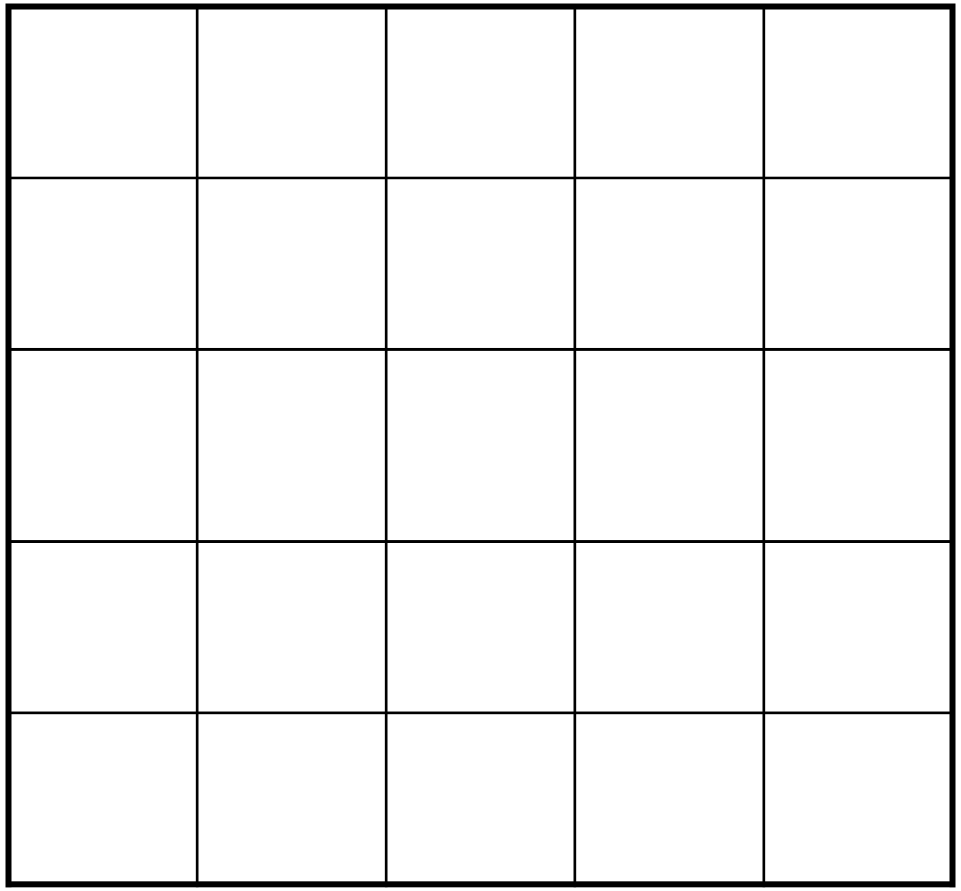
Introduction to Computer & Lab

Lecture No. 2

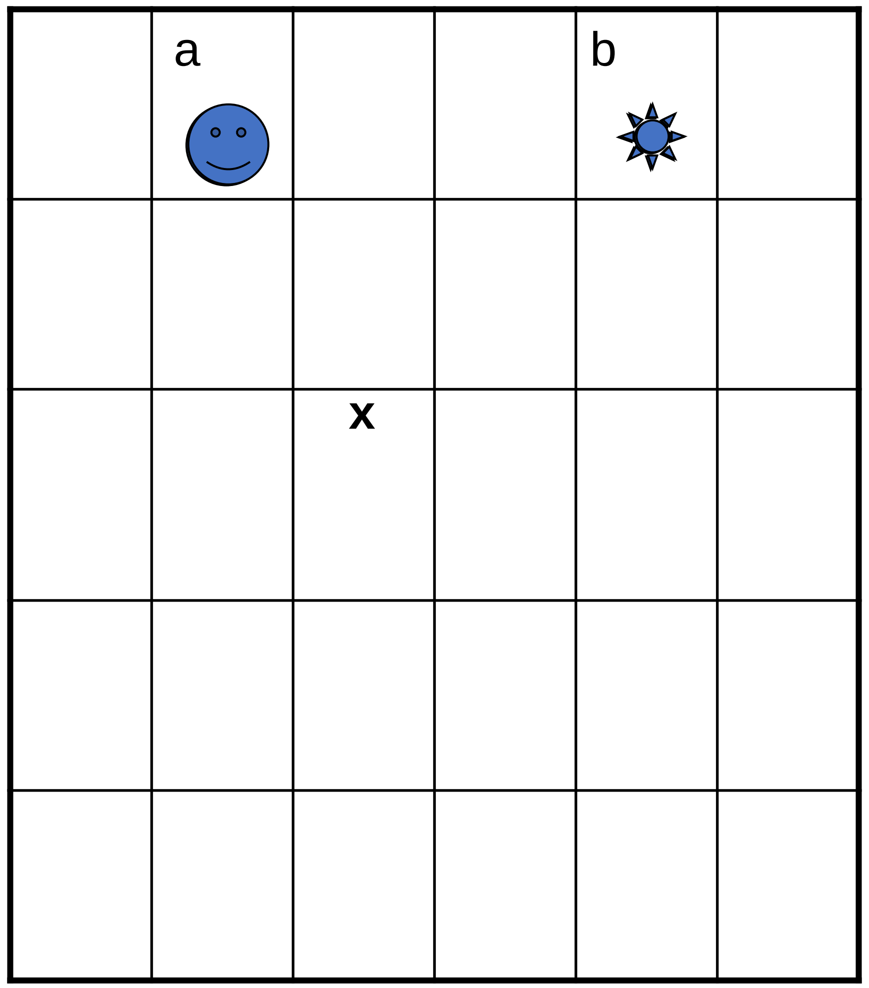
Memory

```
x = 2 + 4;
  = 6;
```



Memory

x = a + b ;



a*(b%c) = a*b%c

?

- No expression on the left hand side of the assignment
- Integer division truncates fractional part
- Liberal use of brackets/parenthesis

Code

```
#include <iostream.h>
main ( )
{
        int number;
        int digit;
        cout << “Please enter a 4 digit integer : ”;
        cin >> number;
        digit = number %10;
        cout <<“The digit is: “ << digit << ‘₩n’;
        number = number / 10;
        digit = number % 10;
        cout <<“The digit is: “ << digit << ‘₩n’;
        number = number / 10;
        digit = number % 10;
        cout <<“The digit is: “ << digit << ‘₩n’;
        number = number / 10;
        digit = number % 10;
        cout <<“The digit is: “ << digit;


}
```

Decision

If Statement

```
If condition is true
  statements

If  Ali's height is greater then 6 feet
Then
  Ali can become a member of the Basket
  Ball team
```

If Statement in C

```
If (condition)
  statement ;
```

If Statement in C

```
If ( condition )
{
        statement1 ;
        statement2 ;
                :
}
```

Relational Operators

<	**less than**
<=	**less than or equal to**
==	**equal to**
>=	**greater than or equal to**
>	**greater than**
!=	**not equal to**

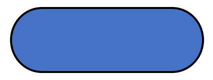
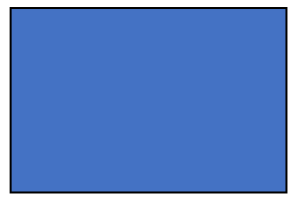
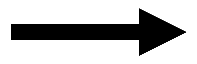
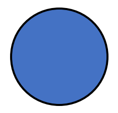
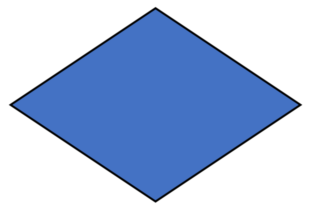
Relational Operators

```
a != b;

X = 0;
X == 0;
```

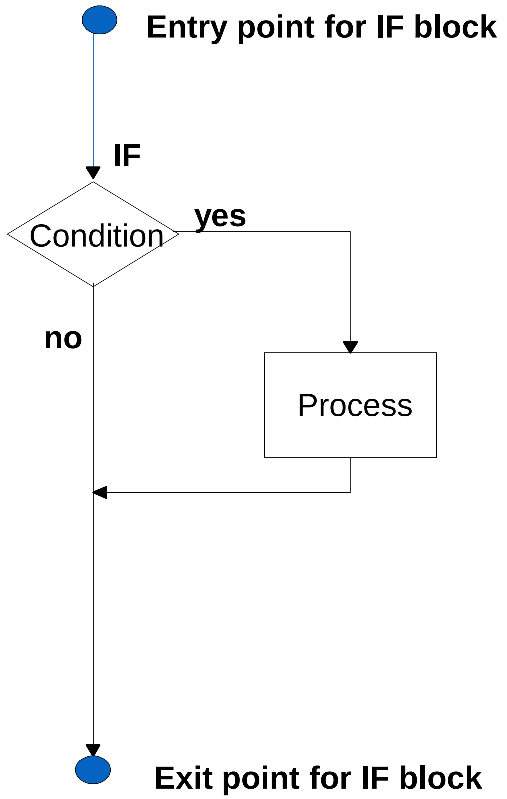
Example

```
#include <iostream.h>
main ( )
{
      int AmirAge, AmaraAge;
      AmirAge = 0;
      AmaraAge = 0;

      cout<<“Please enter Amir’s age”;
      cin >> AmirAge;
      cout<<“Please enter Amara’s age”;
      cin >> AmaraAge;

      if AmirAge > AmaraAge)
      {
         cout << “₩n”<< “Amir’s age is greater than Amara’s age” ;
      }
}
```

Flow Chart Symbols

Start or stop	
Process	
Flow line	
Continuation mark	
Decision	

Flow Chart for if statement



Logical Operators

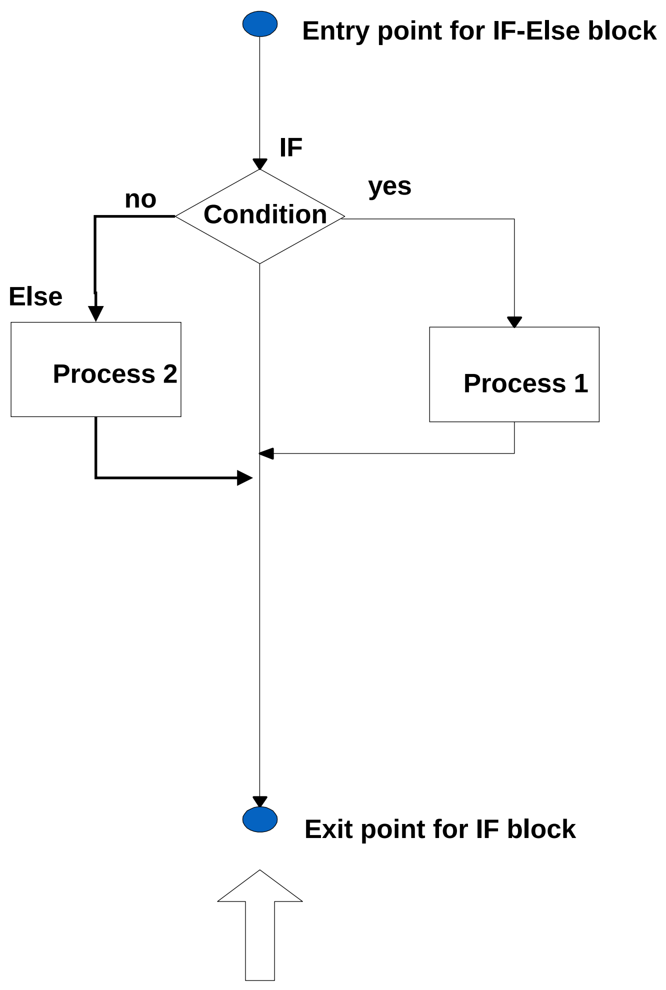
If a is greater than b
AND c is greater than d

```
if(a > b && c> d)
if(age > 18 || height > 5)
if(!(age > 18) || height < 5)
```

if-else

```
if (condition)
{
   statement ;
         -
         -
}
else
{
   statement ;
         -
         -
}
```

if-else


Entry point for IF-Else block
IF
no
yes
Condition
Else
Process 2
Process 1
Exit point for IF block

Example

Code

```
if (AmirAge > AmaraAge)
{
    cout<< "Amir is older than Amara" ;
}

if (AmirAge < AmaraAge)
{
    cout<< "Amir is younger than
Amara" ;
}
```

VS

Code

```
if AmirAge >  AmaraAge)
{
    cout<< "Amir is older than Amara" ;
}
else
{
    cout<<"Amir is not older than Amara" ;
           "Amir is younger than Amara";
}
```

Loop - Repetition structure

Example

```
int sum ;
sum = 1+2+3+4+5+........+10 ;
cout << sum ;
```

Find the Sum of the first 100 Integer starting from 1

?

while
for
do-while

Example

```
int sum , number ;
sum = 0 ;
number = 1 ;
while ( number <= 1000 )
{
        sum   =   sum   +   number ;
        number   =   number + 1 ;
}
cout << ″ The sum of the first 1000 integer starting from 1 is ″ << sum ;
```

Example

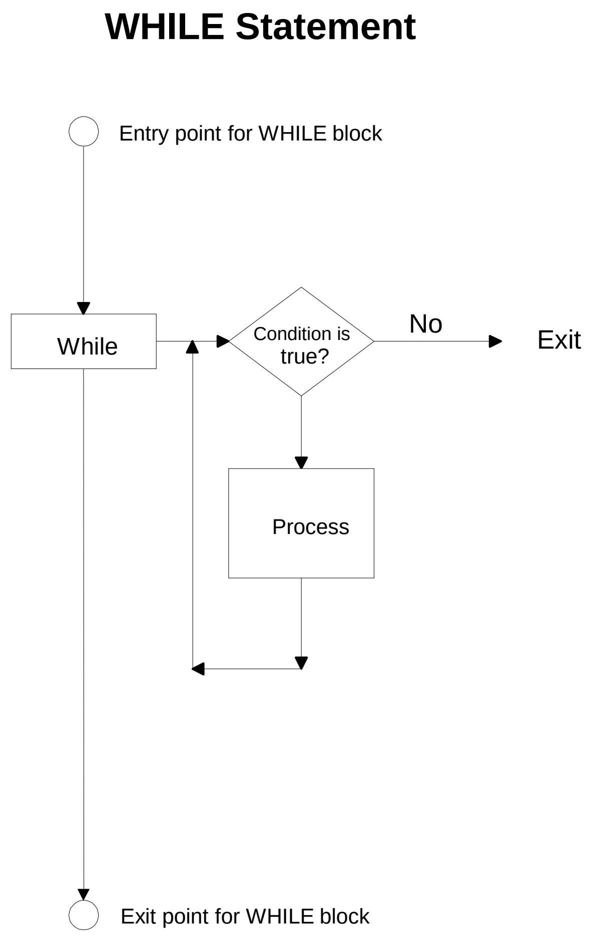
```
int sum, number , UpperLimit ;
sum = 0 ;
number = 1 ;
cout << “ Please enter the upper limit for which you want the sum ” ;
cin >> UpperLimi t;
while (number <= UpperLimit)
{
        sum = sum + number ;
        number = number +1 ;
}
cout << “ The sum of the first ” << UpperLimit << “ integer is ” << sum ;
```

```
if ( number % 2 == 0 )
{
      sum = sum + number ;
      number = number + 1 ;
}
```

Example

```
sum = 0;
number = 1;
cout << " Please enter the upper limit for which you want the sum ";
cin >> UpperLimit;
while (number <= UpperLimit)
{
   if (number % 2 == 0)
   {
            sum = sum + number;
            number = number + 1;
   }
}
cout << " The sum of all even integer between 1 and " << UpperLimit << " is" << sum;
```

Flow Chart for While Construct



Factorial Definition

n! = n*(n-1)*(n-2)*(n-3)………..*3*2*1

Example: Factorial

```
#include <iostream.h>
main ( )
{
   int number ;
   int factorial ;
   factorial = 1 ;
   cout << “Enter the number of Factorial” ;
   cin >> number ;
   while ( number >= 1 )
            {
                        factorial = factorial * number ;
                        number = number – 1 ;
            }
   cout << “Factorial is” << factorial ;
}
```

Property of While Statement

It executes zero or more times

do-while

Do while loop execute one or more times

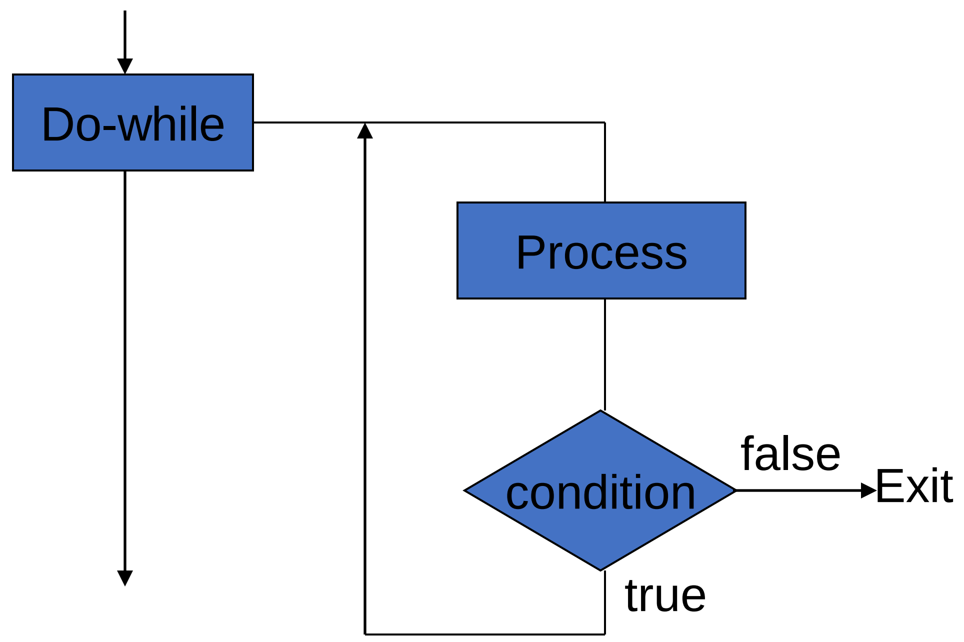
Syntax of do-while loop

```
do
 {
      statements ;

 }
while ( condition ) ;
```

Example-Guessing game

```
char c ;
int tryNum = 1 ;
do
{
  cout << "Please enter your guess by pressing a character key from a to z “ ;
  cin >> c ;
  if ( c == 'z’ )
   {
          cout << "Congratulations! you guessed the right answer“ ;
          tryNum = 6 ;
  }
  else
          tryNum  = tryNum + 1 ;
} while ( tryNum <= 5 ) ;
```

Flow chart for do-while loop



Relational Operators

```
char c ;
 int tryNum , maxTries ;
 tryNum = 1 ;
 maxTries = 5 ;
 cout << "Guess the alphabet between a to z “ ;
 cin >> c ;
 while ( ( tryNum <= maxTries ) && ( c! = ‘z‘ ) )
 {
       cout << "Guess the alphabet between a to z “ ;
      cin >> c ;
      tryNum = tryNum + 1 ;
 }
```

for Loop

For loop

```
for ( initialization condition ; termination condition ; increment condition )
{
   statement ( s ) ;
}
```

Example

```
int counter ;
for( counter = 0 ; counter < 10 ; counter = counter + 1 )
        cout << counter;
```

Output

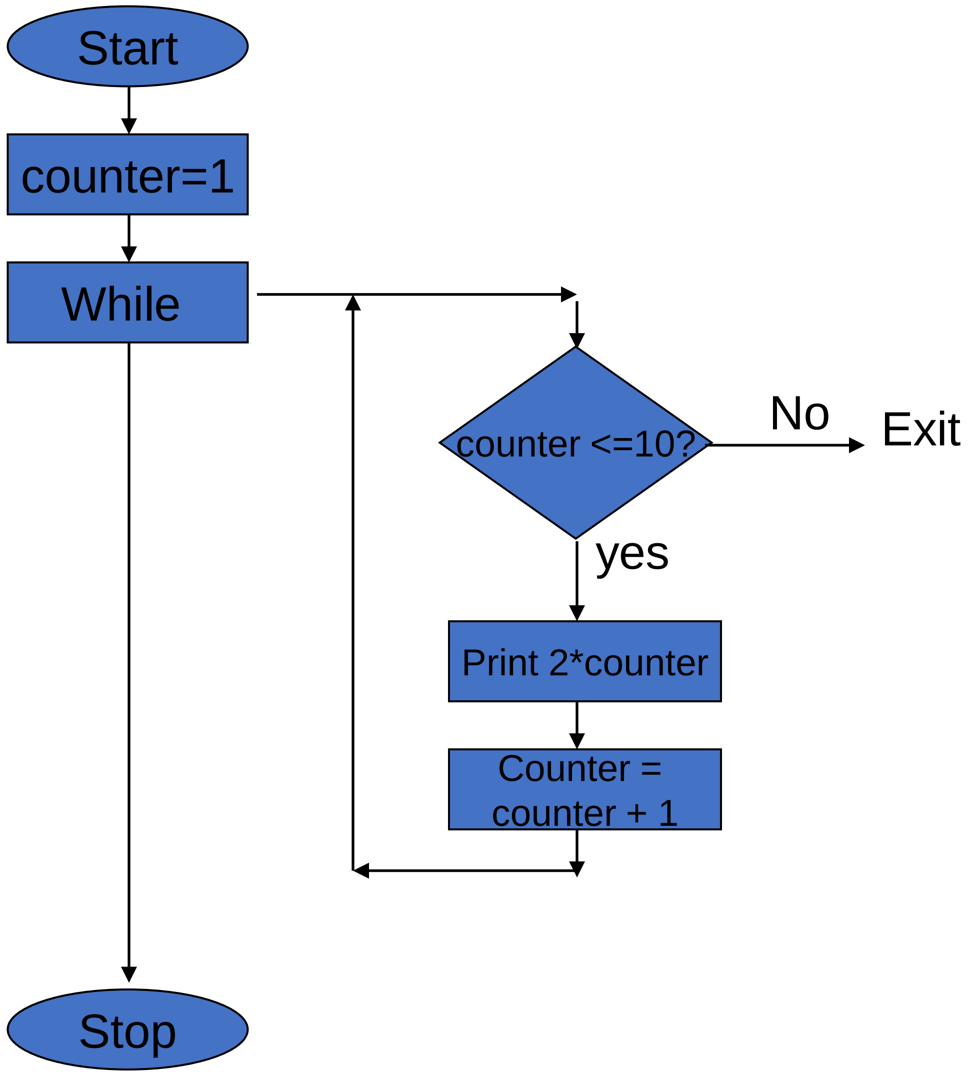
```
0123456789
```

Example - Calculate Table for 2

```
#include <iostream.h>
main ( )
{
   int counter ;
   for ( counter = 1 ; counter <= 10 ; counter = counter + 1 )
   {
        cout << "2 x " << counter << " = " << 2* counter << "₩n“ ;
    }
}
```

```
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
:
:
2 x 10 = 20
```

Flow chart for the ‘Table’ example



Example: Calculate Table- Enhanced

```
#include <iostream.h>
main ( )
{
    int number ;
    int maxMultiplier ;
    int counter ;
    maxMultiplier = 10 ;
    cout << " Please enter the number for which you wish to construct the table “ ;
    cin >> number ;
    for ( counter = 1 ; counter <= maxMultiplier ; counter = counter + 1 )
    {
                cout << number <<" x " << counter<< " = " <<  number * counter << "₩n“ ;
    }
}
```

- Always think re-use
- Don’t use explicit constants

Increment operator
++

- **counter ++ ;**

  same as
- **counter = counter + 1;**

Decrement operator --

- **counter -- ;**

  same as
- **counter = counter - 1**

Compound Assignment Operators

***operator*=**

+=

- counter += 3 ;

  same as
- counter = counter + 3 ;

-=

- counter -= 5 ;

  same as
- counter = counter – 5 ;

%=

- x %= 2 ;

  same as

- x = x % 2 ;

*=

- x*=2;

  same as

- x = x * 2;

/=

- x /= 2;

  same as

- x = x / 2’

Example: Program to calculate the average marks of class

```
int sum;
int students ;
int average ;
sum = 0 ;
students = 0 ;
do
{
            cin >> grade ;
            sum += grade ;
            students ++ ;
}
while (grade >= 0) ;
average = sum / students ;
cout << average ;
```

A logical flaw in the code (HW)

Multi-way decision

if Statements

```
if  ( grade ==‘A’ )
   cout << “ Excellent ” ;
if  ( grade ==‘B’ )
   cout << “ Very Good ” ;
if  ( grade ==‘C’ )
   cout << “ Good ” ;
if  ( grade ==‘D’ )
   cout << “ Poor ” ;
if  ( grade ==‘F’ )
   cout << “ Fail ” ;
```

if else

```
if  ( grade ==‘A’ )
  cout << “ Excellent ” ;
else
  if  ( grade ==‘B’ )
        cout << “ Very Good ” ;
else
  if  ( grade ==‘C’ )
        cout << “ Good ” ;
else
  if  ( grade ==‘D’ )
        cout << “ Poor ” ;
else
        cout << “ Fail” ;
```

To complete the all logical possibilities.
Is it enough?

if else

```
if ( grade == ‘A’ )
  cout << “ Excellent ” ;
else if ( grade == ‘B’ )
  …
else if …
  …
else …
```

switch statement

switch statements

```
switch ( variable name )
  {
        case 'a' :
                statements;
        case 'b' :
                statements;
        case 'c' :
                 statements;
        …
  }
```

switch statements

```
switch ( grade)
  {
        case ‘A’ :
                cout << “ Excellent ” ;
        case ‘B’ :
                cout << “ Very Good ” ;
        case ‘C’ :
                …
        …
  }
```

switch statements

```
case ‘A’ :
        cout << ” Excellent ” ;
        …
        …
```

Example

```
switch ( grade)
{
        case ‘A’ :
                cout << “ Excellent ” ;
        case ‘B’ :
                cout << “ Very Good ” ;
        case ‘C’ :

                cout << “Good ” ;

        case ‘D’ :
                cout << “ Poor ” ;
        case ‘F’ :
                cout << “ Fail ” ;

}
```

break;

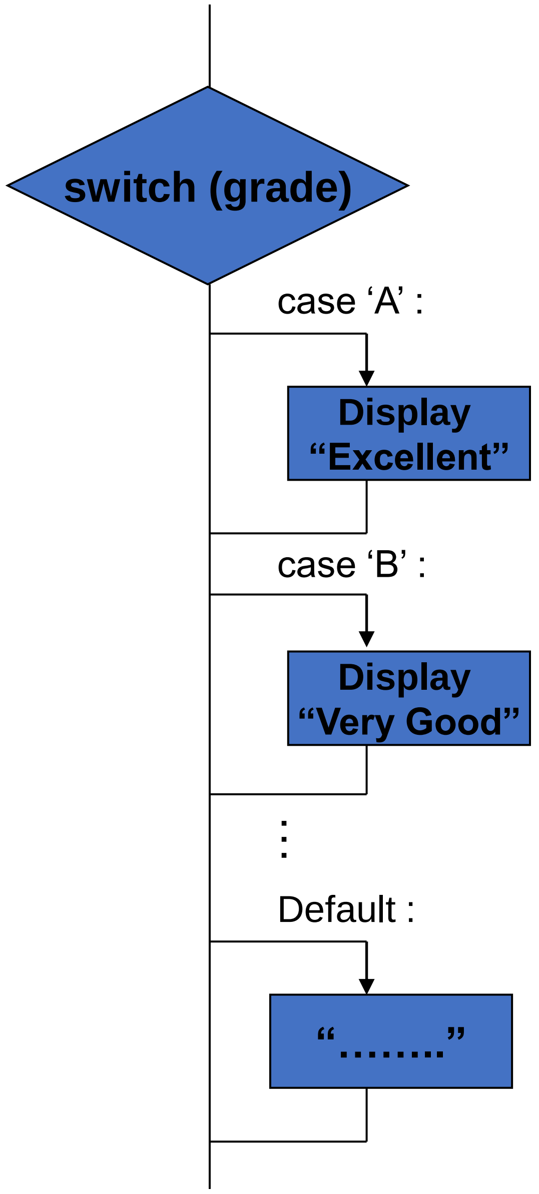
Example

```
switch ( grade )
{
          case ‘A’ :
                    cout << “ Excellent ” ;
                    break ;
          case ‘B’ :
                    cout << “ Very Good ” ;
                     break ;
          case ‘C’ :
                    cout << “Good ” ;
                     break ;
          case ‘D’ :
                    cout << “ Poor ” ;
                     break ;
          case ‘F’ :
                    cout << “ Fail ” ;
                     break ;
}
```

default :

```
default :
  cout << " Please Enter Grade from
  'A' to 'D' or  'F'  " ;
```

Flow Chart of switch statement



```
if ( amount > 2335.09 )
    statements ;
```

Whole Number

- **short**
- **int**
- **long**

```
case ‘A’ :
case ‘ 300 ‘ :
case ‘ f ‘ :
```

break ;

```
if (c == ‘z’ )

{
   cout << “ Great ! You have made the correct guess “ ;
   break ;
}
```

continue ;

continue

```
while (trynum <= 5 )
{
        ....
        continue ;
        ....
        ....

}
```

continue in ‘for’ loop

```
for ( counter = 0 ;counter <= 10 ; counter ++ )
  {
          …….
          continue ;
  }
```

What have we done till now …

- Sequential Statements
- Decisions
    - if , if else , switch
- Loops
    - while , do while , for

goto

Unconditional Branch of Execution

Structured Programming

- Sequences
- Decisions
- Loops

- Minimize the use of break
- Minimize the use of continue
- Never use goto

Guidelines for structured programming

- **Modular**
- **Single entry - single exit**