CSC2212
C++ Programming

Control Structures

Lab three

Flow-Control Statements

These are mechanisms use to change the order of sequence flow of a program using branching and looping concepts

Relational Operators

> Greater than
< Less than
>= Greater than or equal to
<= Less than or equal to
== Equal to
!= Not equal to

These relational Operators are use in controlling the flow of a program

Out come of each comparison is either 1 to represent true or 0 when it is false.

Precedence of Relational Operators

Relational operators also have a precedence order among themselves..

> >= < <= have higher priority
== != have lower priority

Example: c == a > b
a > b will be evaluated first
c == (a > b) will be evaluated second

Flow-Control Statements

Group into two categories:

1. Decision statement
2. Looping statements

1- Decision Statements

Are statement that test a conditions and take action according to the outcome of the test.

There are three statements for altering the flow of program in C++

i. if…Then

ii. if…Then…Else

iii. Select Case

(i) If… Statements

The if statement can cause other statements to execute only under certain conditions.

The **if…** statement tests a given condition.
If the condition is ***True***, the program executes the statement (or set of statement inside curly braces) that follow.

If the condition is ***false*** it will simply skip the entire statement (or set of statement inside curly braces) that follow and execute the next statement after the curly braces.

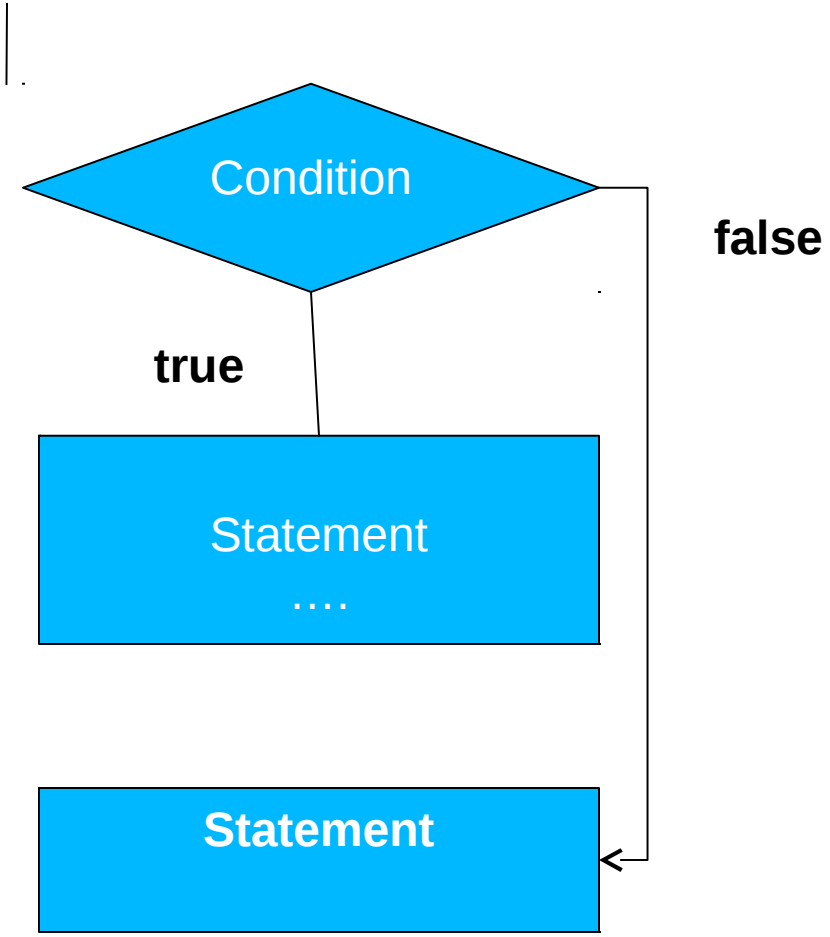
(i) If… Statements

```
If( condition ){

   statement

}

Statement
```



(i) If… ***Statements*(example)**

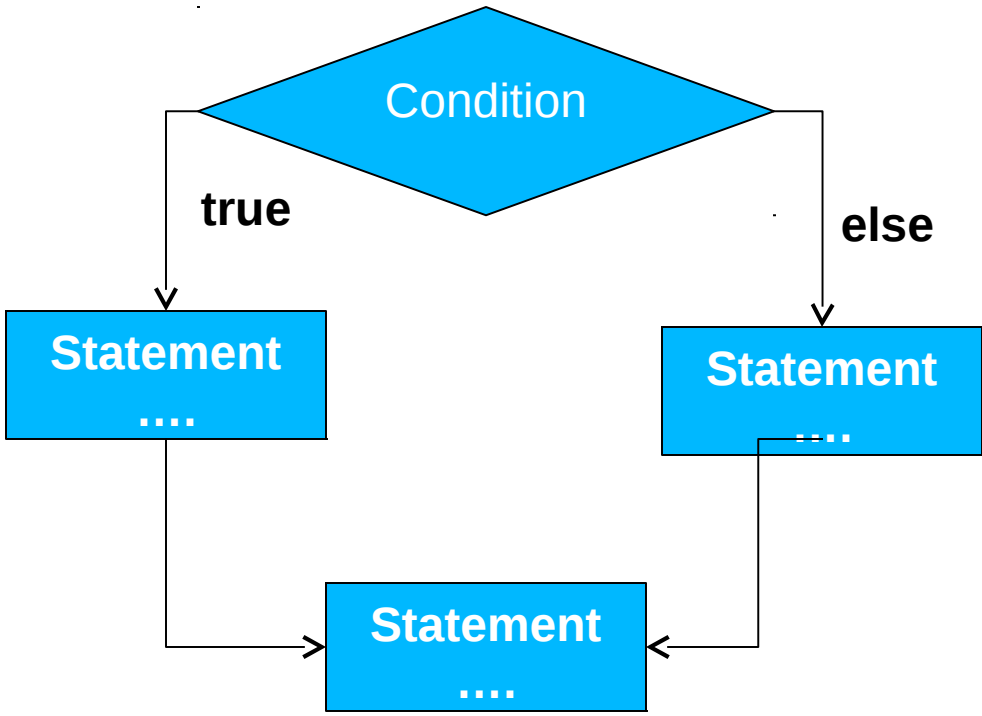
```
If(  temp > 32 ){
     switch off
}
```

(ii) If…else Statements

Executes one block of statements if the condition is True and another block of statements if the condition is False.

Syntax:

```
If( condition ){
    statement . . .
}
Else{
    statement . . .
}
```



(b) If…Then…Else Statements(example 1)

```
If( temp > 32 ){
     turn off
}
Else{
     increase
temp
}
```

(b) If…elseif Statements

The if...else if statement is a chain of if statements. They perform their tests, one after the other, until one of them is found to be true.

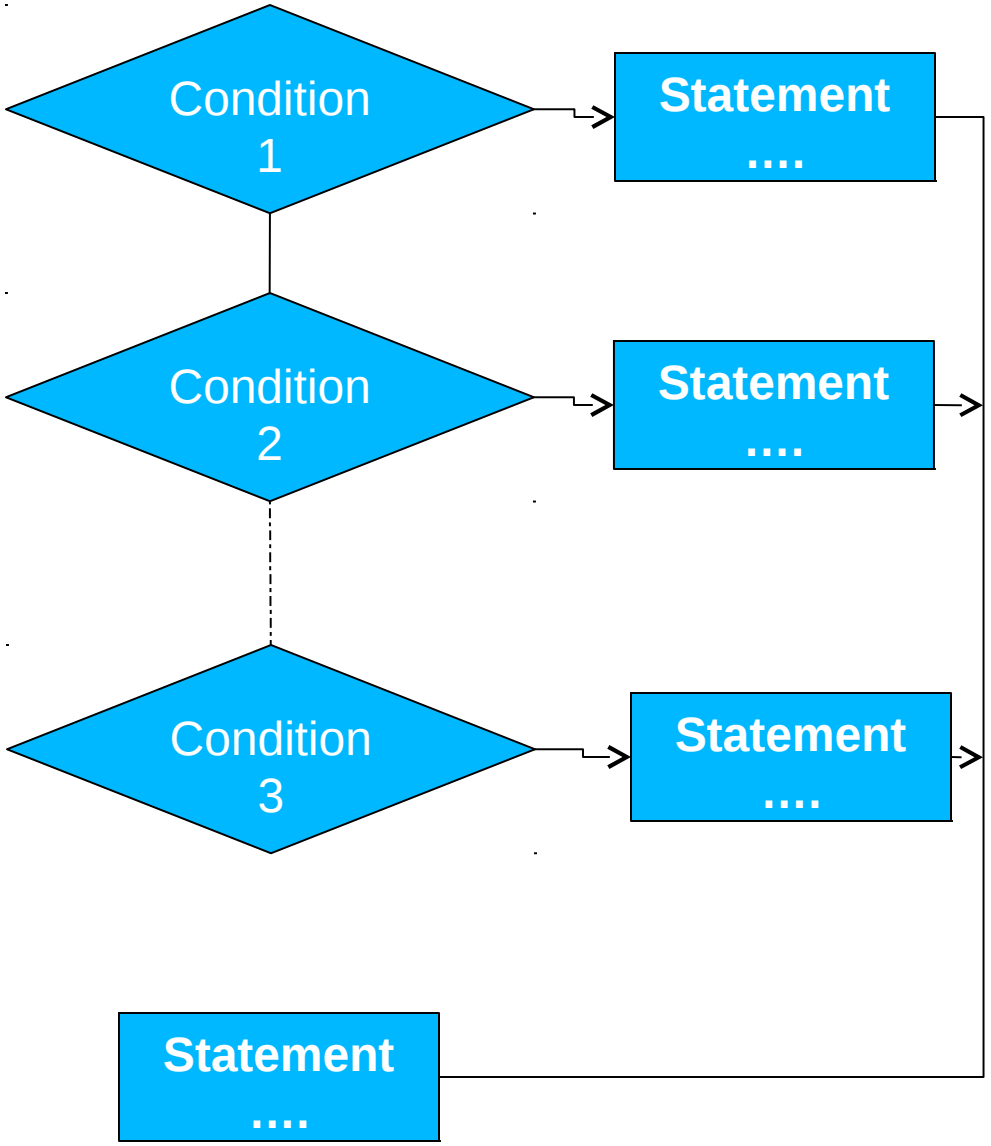
If the condition is False, Visual Basic ignores the first block of statements and executes the block following the Else keyword.

Many *else if* clauses can be included. The conditions are evaluated from the top, and if one of them is True, the corresponding block of statements is executed.

The final else clause, which is optional, will be executed if none of the previous expressions is True.

(b) If…else if Statements

```
if (condition 1)
{ statement
 }
else if (condition 2)
{ statement
}
      . . .
else{
statement
}
```



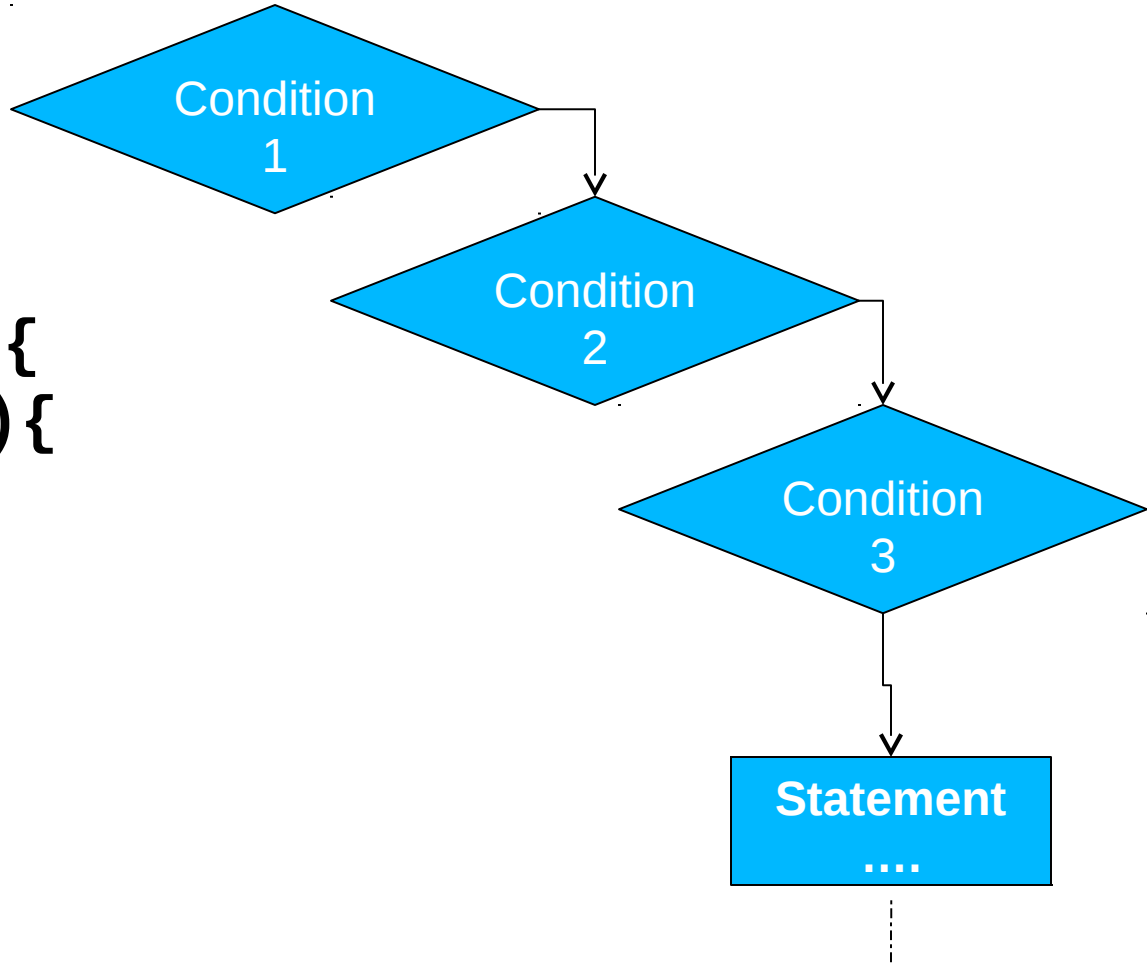
Nested if statements

To test more than one condition, an if statement can be nested inside another if statement.

```
If(condition1){
   if( condition2 ){
      if( conditon3){

         statement

         }
      }
   }
```



Logical Operators

Logical operators connect two or more relational expressions into one or reverse the logic of an expression

&&	**AND**	Connects two expressions into one. Both expressions must be true for the overall expression to be true.
||	**OR**	Connects two expressions into one. One or both expressions must be true for the overall expression to be true. It is only necessary for one to be true, and it does not matter which.
!	**NOT**	Reverses the “truth” of an expression. It makes a true expression false, and a false expression true.

The Conditional Operator

- Can use to create short ***if … else*** statements
- Format:
  `condition ? Statement1 : statement2;`
- Test the given condition if true execute statement1 if false execute statement2
- Example:
  - **`( x > 5 )? x -= 3 : x += 2`**

The Conditional Operator

- Can use to create short ***if … else*** statements
- Format:
  `condition ? Statement1 : statement2;`
- Test the given condition if true execute statement1 if false execute statement2
- Example:
  - **`( x > 5 )? x -= 3 : x += 2`**

(iii) Switch Statement

- Used to select one statements from many several alternatives
- Is an alternative to the multiple ***if…else if*** statements
- It makes code compact, easier to read and maintain
- Syntax

```
  switch ( condition)
{
case expresion1: statement 1;break;
case expresion2: statement 2; break;
                 ...
case expresion3: statement n; break;
default:   last statement;
}
```

(iii) Switch Statement

- **`Condition`**: it has to evaluates to an integer value

- **`expression1 up to the last expression:`** must be constant integer type expressions and must be unique in the switch statement

- **`default`** is optional but recommended

(iii) Switch Statement

```
switch ( number )
{
  case 1: cout << “One";
            break;
  case 2: cout << “Two";
            break;
  default : cout << “Others";
}
```

LOOP

Use to indicate part of a program that may need to be repeated more than ones

i.while loop

ii.do-while loop

iii.for loop

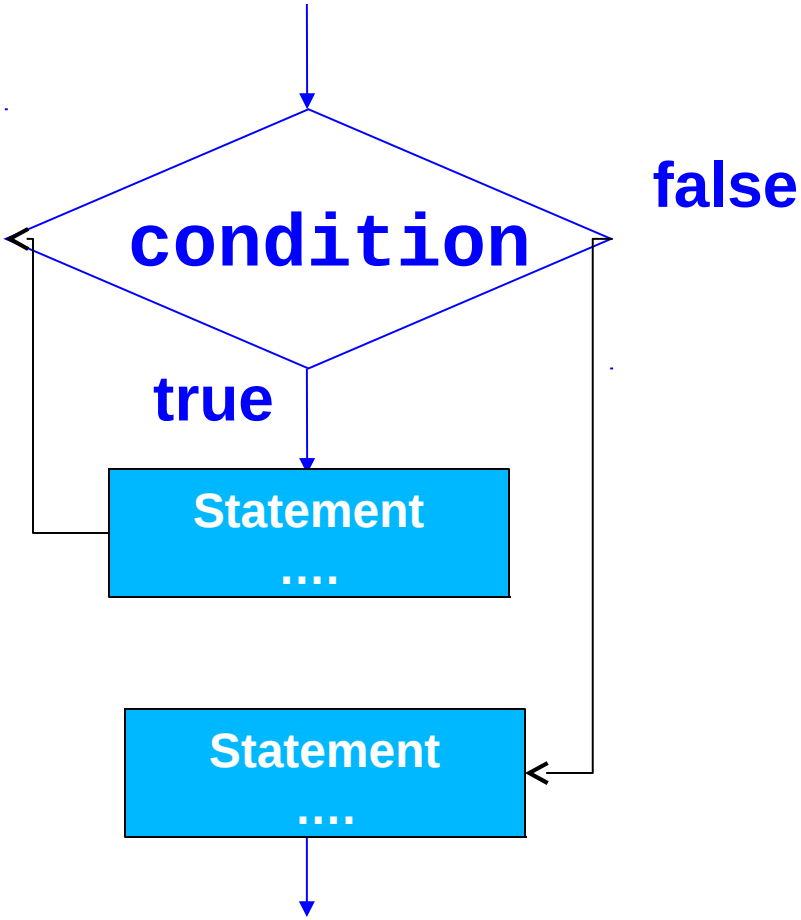
While Loop

while loop format:

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

condition is evaluated, if it is true, the statement(s) are executed, and then condition is evaluated again if it is false, the program exit the loop

While Loop



While Loop(example)

```
int val = 1;
while( val <= 9 )
{
    cout << val << "   ";
    val = val + 1;
}
```

do while Loop

do-while: a post test loop (***condition*** is evaluated after the loop executes)

Format:

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

for Loop

- ***For loop*** requires that the number of times a statements would be repeated is known.
- Useful for counter-controlled loop
- Pretest loop that executes zero or more times

Format:

```
for(initialization; condition; update)
{
   statements to be repeated;
}
```

for Loop(example)

```
int sum = 0, num;

for (num = 1; num <= 10; num = num +2){
        sum += num;

  }

cout << "Sum of even numbers from 1 –
100 is “ << sum << endl;
```

for Loop

- If ***condition*** is false the first time it is evaluated, the body of the loop will not be executed.
- The update expression can increment or decrement by any amount.
- Variables used in the initialization section should not be modified in the body of the loop

Nested Loops

- A loop can be inserted inside the body of another loop

- Example:

```
for (row = 1; row <= 3; row++)
{
  for (col = 1; col <= 3; col++)
  {
     cout << row * col << endl;
  }
}
```

Breaking Out of a Loop

- Can use ***`break`*** to terminate execution of a loop
- Use sparingly if at all – makes code harder to understand
- When used in an inner loop, terminates that loop only and returns to the outer loop

Breaking Out of a Loop

- Can use continue to go to end of loop and prepare for next repetition
  - while and do-while loops go to test and repeat the loop if test condition is true
  - for loop goes to update step, then tests, and repeats loop if test condition is true
- Use sparingly – like break, can make program logic hard to follow

Exercise One

- Write a program to calculate bonus for a customer base on his spending, If the customer spend more than 10,000 can get 10% bonus, 15% on spending more than 15,000, 20% on spending more than 20,000 also eligible for 25% on spending more than 30,000.

Exercise One

- Write a program to generate number 1 to 12
- Modify the program to generate multiplication table in 12 x 12 table.

- WAP to check whether the entered number is palindrome or not
- WAP that will reverse sequences of number.
- WAPs to print prime numbers out of 1000 number.
- WAP that will compute the following patterns of numbers.
- 1
- 1 2
- 1 2 3
- 1 2 3 4
- 1   2 3 4 5
- .
- .
- 1 2 3 4 5 . . . . n