Computer I

Lecture No. 5
Control Flow

Al-Mustaqbal University College of Engineering & Technology
Biomedical Engineering Department
MSc. in Computer Engineering: Hamza Waleed Hamza

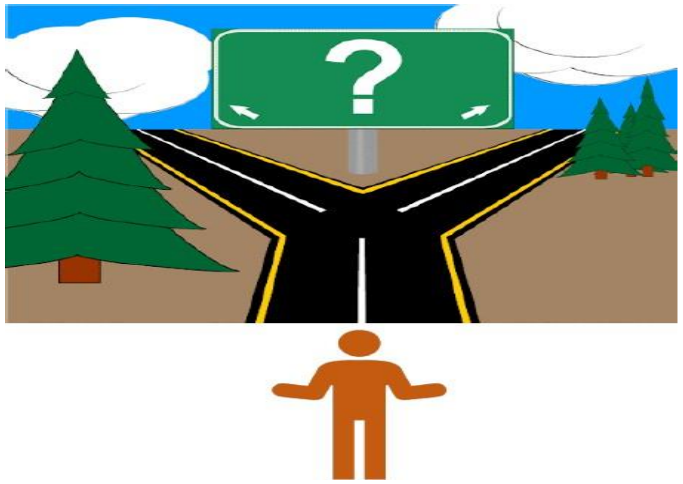
Conditions

- Conditions are very important to understand the flow-control of the code being executed.
- Conditions are used to make a decision (choose between two or more alternatives based on the condition)

  - For example, conditions become very essential to use when facing different directions, so based on the condition should choose the thing to be executed.

Use the if statement

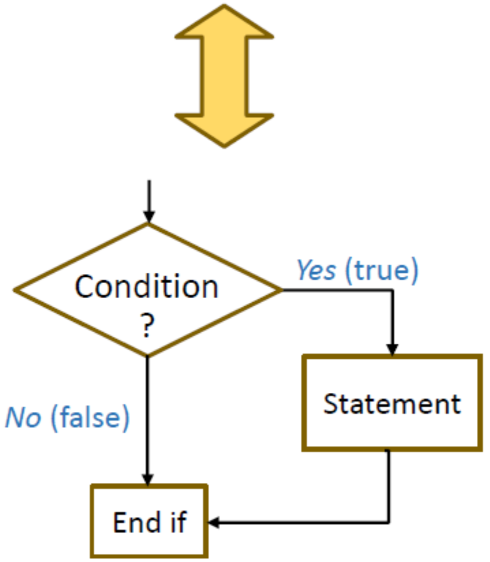
If (condition) then run this statement;
Else
run the other statement;



Conditions

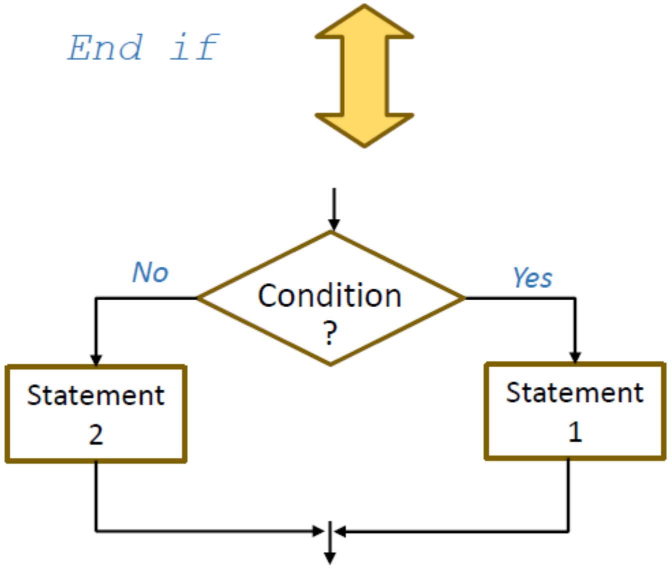
1 ***if …. then***

```
If (condition) then
 statement(s)

End if
```



2 ***if …. then …. else***

```
If (condition) then
 statement 1
Else
 statement 2

End if
```



Nested if structure

3 ***Nested structure of***
if .... then

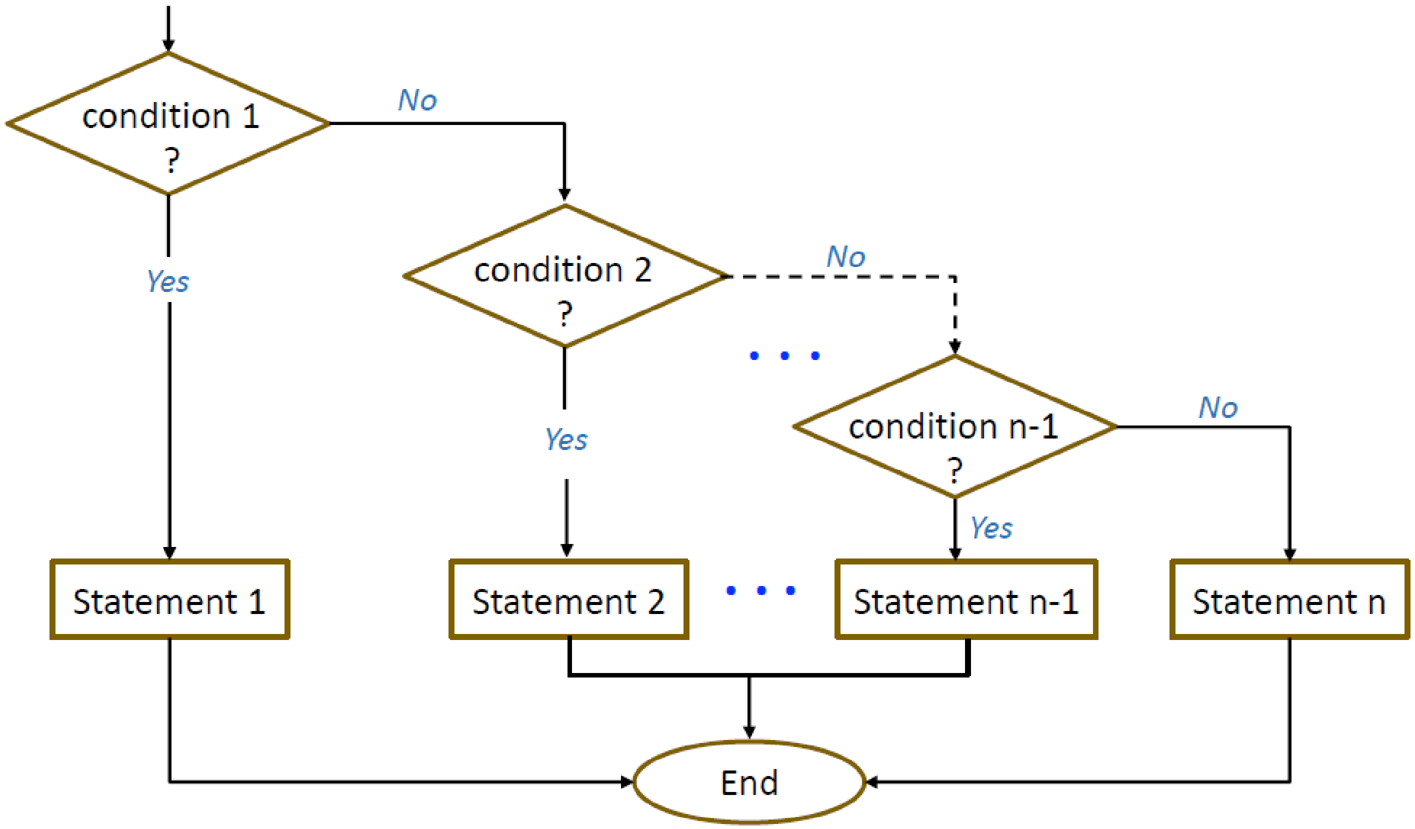
```
If (condition 1) then
 statement 1

If (condition 2) then
 statement 2

If (condition 3) then
 statement 3

Else
   statement 4

End if
End if
End if
```

if .... then .... elseif

```
If (condition 1) then
 statement 1

Elseif (condition 2) then
 statement 2

Elseif (condition 3) then
 statement 3

Else
  statement 4

End if
```

if … then … elseif – Flowchart

- *if .... then .... elseif* Flowchart



Example 1

```
#include <iostream>
using namespace std;
int main()
{
 int number;
cout << "Enter an integer: ";
cin >> number;
if ( number > 0)
{
cout << "You entered a positive integer: " << number ;
 }
return 0;

}
```

Example 2

```
#include <iostream>
 using namespace std;
 int main()
{
 int number;
 cout << "Enter an integer: "; cin >> number;
if ( number >= 0) {
 cout << "You entered a positive integer: " << number; }
 else {
cout << "You entered a negative integer: " << number; }
}
```

LOOPS

WHY DO WE NEED LOOPS ? ? ?

- There may be a situation, when you need to execute a block of code several number of times.
- In general statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.
- A loop statement allows us to execute a statement or group of statements multiple times

TYPES OF LOOPS :

- **WHILE LOOP**
- **FOR LOOP**
- **DO-WHILE LOOP**
- **NESTED LOOP**

LOOPS => WHILE LOOP

A **while** loop statement repeatedly executes a target statement as long as a given condition is true.

Syntax:

The syntax of a while loop in C is:

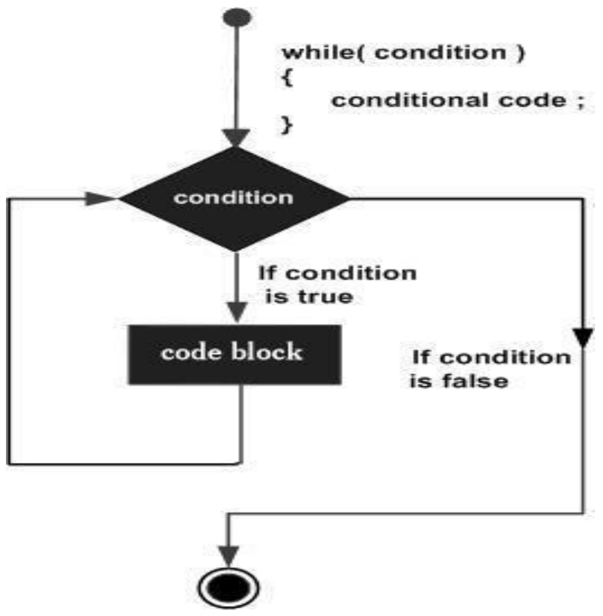
```
while(condition)
 {
    statement(s);
 }
```

LOOPS => WHILE LOOP

- Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any non-zero value. The loop iterates while the condition is true.

- When the condition becomes false, program control passes to the line immediately following the loop

FLOW DAIGRAM



LOOPS => WHILE LOOP

EXAMPLE :

```
#include <iostream>
#include<stdlib.h> int main
()
{
int a = 10; while( a <
20 )
 {
     printf(“value of a:%d \n”, a); a++;
  }
}
```

When the above code is compiled and executed, it produces the following result:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

LOOPS => FOR LOOP

FOR LOOP:

A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax:

The syntax of a for loop in C is:

```
 for ( init; condition; increment )
  {
    statement(s);
   }
```

LOOPS => FOR LOOP

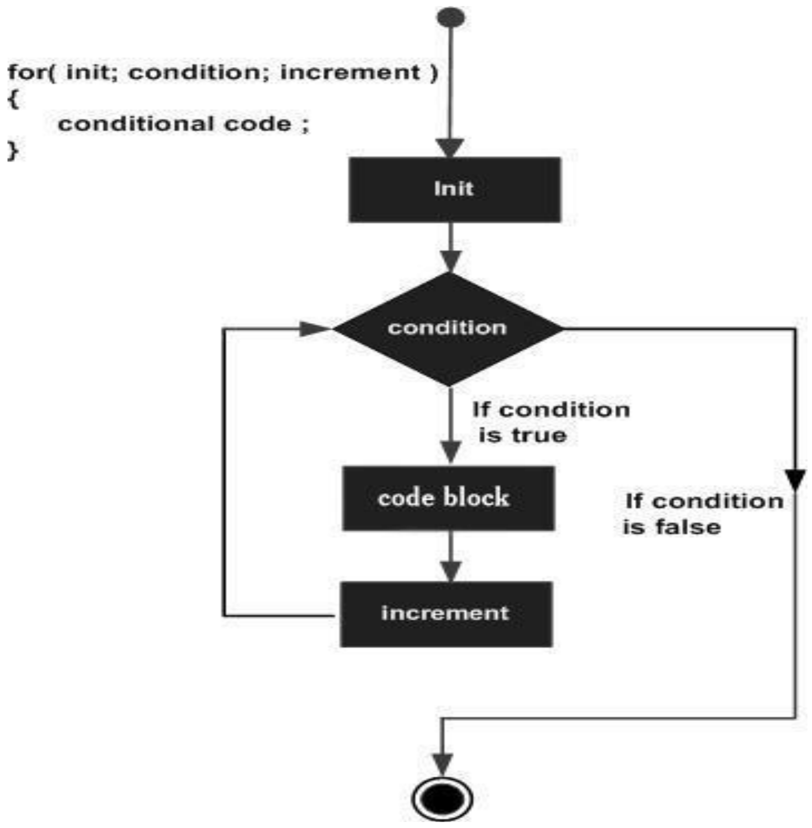
- The **init** step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
- Next, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement just after the for loop.
- After the body of the for loop executes, the flow of control jumps back up to the increment statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.
- The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the for loop terminates.

LOOPS => FOR LOOP

Example:

```
#include<stdlib.h>
#include <iostream>
int main ()
{
for( int a = 10; a < 20; a = a + 1 )
  {
    printf("value of a: %d \n", a);
  }
}
```

Flow Diagram



LOOPS => FOR LOOP

- When the above code is compiled and executed, it produces the following result:

```
value  of  a:  10
value  of  a:  11
value  of  a:  12
value  of  a:  13
value  of  a:  14
value  of  a:  15
value  of  a:  16
value  of  a:  17
value  of  a:  18
value of a:   19
```

LOOPS => DO-WHILE LOOP

DO-WHILE LOOP:

- Unlike **for** and **while** loops, which test the loop condition at the top of the loop, the **do...while** loop checks its condition at the bottom of the loop.
- A **do...while** loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

Syntax:

The syntax of a do...while loop in C is:

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

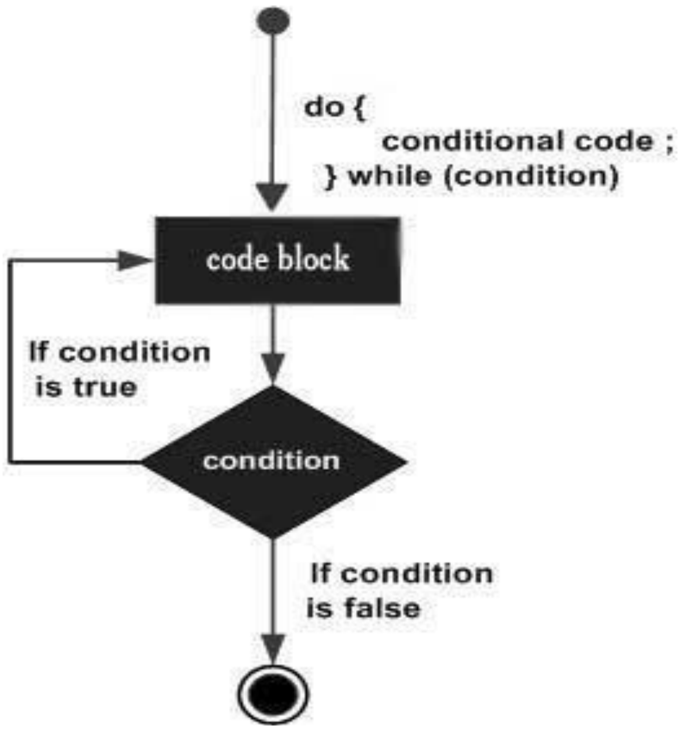
Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop execute once before the condition is tested.

LOOPS => DO-WHILE LOOP

Example:

```
#include<stdlib.h>
#include <iostream>
  int main ()
 {
    int a = 10;
do
    {
      printf( "value of a: %d\n " ,a);
      a = a + 1;
     }while( a < 20 );
}
```

Flow Diagram



LOOPS => DO-WHILE LOOP

- When the above code is compiled and executed, it produces the following result:

```
value  of  a:  10
value  of  a:  11
value  of  a:  12
value  of  a:  13
value  of  a:  14
value  of  a:  15
value  of  a:  16
value  of  a:  17
value  of  a:  18
value of a:    19
```

LOOPS => NESTED LOOP

- **NESTED LOOPS :**
- **A loop can be nested inside of another loop.**
- **Syntax:**

The syntax for a **nested for loop** statement in C is as follows: for (init; condition; increment)

```
 {
   for ( init; condition; increment )
     {
       statement(s);
     }
   statement(s);
   // you can put more statements.
 }
```

LOOPS => NESTED LOOP

EXAMPLE :

```
 #include<stdlib.h> int main
 ()
 {
   int a=1,b;
   while(a<=3)
    {
      for(b=1;b<=3;b++)
      {
        printf("a = %d , b = %d\n",a,b);
      }
      printf("\n"); a++;
    }
}
```

- When the above code is compiled and executed, it produces the following result:

```
a = 1 , b = 1
a = 1 , b = 2
a = 1 , b = 3

a = 2 , b = 1
a = 2 , b = 2
a = 2 , b = 3

a = 3 , b = 1
a = 3 , b = 2
a = 3 , b = 3
```

Exercise 1: Write a program to check if a number is even or odd.

```
#include <stdio.h>
#include <iostream>
int main() {
  int num;
  printf("Enter an integer: ");
  scanf("%d", Cnum);

  if (num % 2 == 0) {
    printf("%d is an even number.\n", num);
  } else {
    printf("%d is an odd number.\n", num);
  }
}
```

Exercise 2: Write a program that takes an integer input N and prints all even numbers from 1 to N using a for loop.

```
#include <iostream>
using namespace std;
int main() {
  int N;
  cout << "Enter the value of N: "; cin >> N;
  cout << "Even numbers from 1 to " << N << " are: ";
  for (int i = 1; i <= N; i++) {
    if (i % 2 == 0) {
      cout << i << " ";
    }
  }
  cout << endl;
}
```

Exercise 3: Write a program that calculates the sum of the first N natural numbers using a while loop.

```
#include <iostream>
using namespace std;
int main() {
  int N, sum = 0, i = 1;
  cout << "Enter the value of N: "; cin >> N;
  while (i <= N) { sum += i; i++;
  }
  cout << "The sum of the first " << N << " natural numbers is: " << sum <<
  endl;

  return 0;
}
```

Exercise 4: Write a program that prints the multiplication table of a number using a for loop.

```
#include <stdio.h>
int main() {
int num;
  printf("Enter a number to print its multiplication table: "); scanf("%d", Cnum);

  printf("Multiplication table of %d:\n", num); for (int i = 1; i <= 10; i++) {
    printf("%d * %d = %d\n", num, i, num * i);
  }

  return 0;
}
```

Exercise 5: Write a program to find the largest of three numbers using if-else

```
#include <stdio.h>

int main() {
  int num1, num2, num3; printf("Enter
  three numbers: ");
  scanf("%d %d %d", Cnum1, Cnum2, Cnum3);

  if (num1 >= num2 CC num1 >= num3) {
    printf("The largest number is %d\n", num1);
  } else if (num2 >= num1 CC num2 >= num3) {
    printf("The largest number is %d\n", num2);
  } else {
    printf("The largest number is %d\n", num3);
  }
}
```

break Statement

- In C programming, break is used in terminating the loop immediately after it is encountered. The break statement is used with conditional if statement.
- Syntax of break statement

```
break;
```

- Flow Chart Of Break Statement

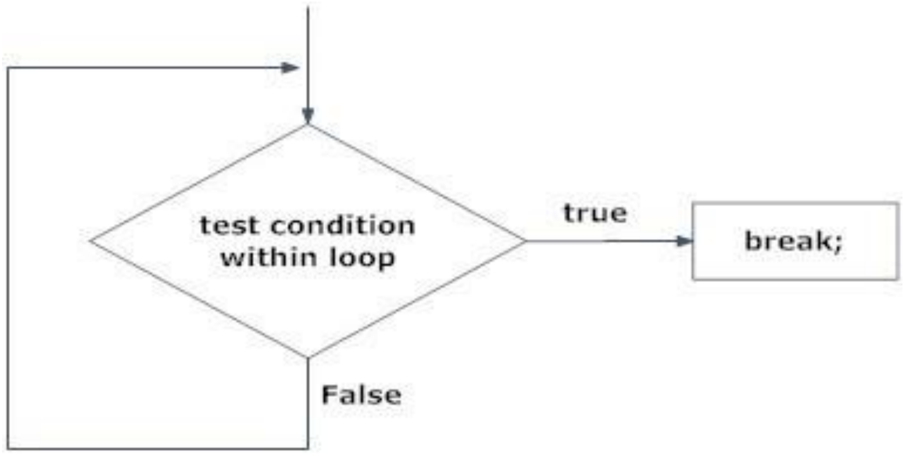


Figure: Flowchart of break statement

Example of break statement

Write a C program to find average of maximum of *n* positive numbers entered by user. But, if the input is negative, display the average(excluding the average of negative input) and end the program.

```
#include <iostream>
using namespace std;
int main() {
    int n;
    float num, sum = 0;
    int count = 0;
    cout << "Enter the maximum number of inputs: ";
    cin >> n;
    for (int i = 1; i <= n; ++i) {
        cout << "Enter number " << i << ": ";
        cin >> num;
        if (num < 0) {
            break; // End input on negative number
        }
        sum += num;
        count++;
    }
    if (count > 0) {
        float average = sum / count;
        cout << "Average = " << average << endl;
    } else {
        cout << "No positive numbers entered." << endl;
    }

    return 0;
}
```

```
Enter the maximum number of inputs: 4
Enter number 1: 1.5
Enter number 2: 12.5
Enter number 3: 7.2
Enter number 4: -1
Average = 7.07
```

Continue Statement

- It is sometimes desirable to skip some statements inside the loop. In such cases, continue statements are used.
- Syntax of continue statement continue;
- Flow Chart Of Continue Statement

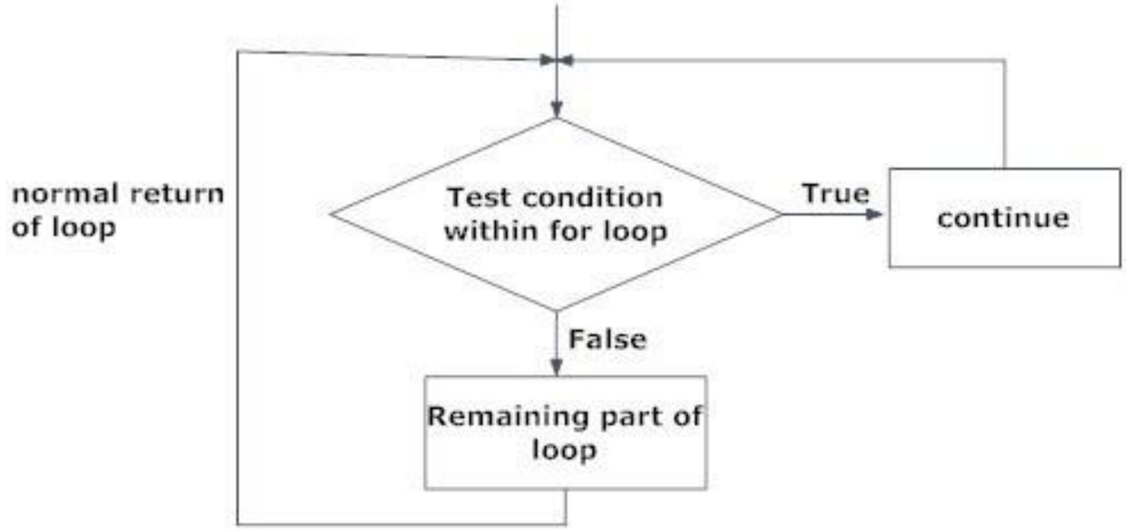


Fig: Flowchart of continue statement

Example of continue statement

Example: Skip Number 3

```
#include <iostream>
using namespace std;

int main() {
    for(int i = 1; i <= 5; i++) {
        if(i == 3) {
            continue;   // Skip number 3
        }
        cout << "Number: " << i << endl;
    }
    return 0;
}
```

Output :

```
Number: 1
Number: 2
Number: 4
Number: 5
```

Switch Statement

- Decision making are needed when, the program encounters the situation to choose a particular statement among many statements. If a programmer has to choose one block of statement among many alternatives, nested if...else can be used but, this makes programming logic complex. This type of problem can be handled in C programming using switch statement.

- Syntax of switch...case

switch (n) { case constant1:

code/s to be executed if n equals to constant1; break; case constant2:

code/s to be executed if n equals to constant2; break; . . . default:

code/s to be executed if n doesn't match to any cases;

}

Example of switch...case statement

Write a program that asks user to select an arithmetic operator('+','-','*' or '/') and two operands and perform the corresponding calculation on the operands.

```
# include <stdio.h> int main()
{
char o;
float num1,num2;
printf("Select an operator either + or - or * or /\n"); scanf("%c",Co);
 printf("Enter two operands: ");
 scanf("%f%f",Cnum1,Cnum2); switch(o) {
 case '+':
printf("%.1f + %.1f = %.1f",num1, num2, num1+num2); break;
case '-':
printf("%.1f - %.1f = %.1f",num1, num2, num1-num2); break;
case '*':
printf("%.1f * %.1f = %.1f",num1, num2, num1*num2); break;
case '/':
printf("%.1f / %.1f = %.1f",num1, num2, num1/num2); break;
default:
printf("Error! operator is not correct"); break; }
}
```

Output: Enter operator either + or - or * or /* Enter two operands: 2.3 4.5 2.3 * 4.5 = 10.3

FlowChartOfSwitch...Case

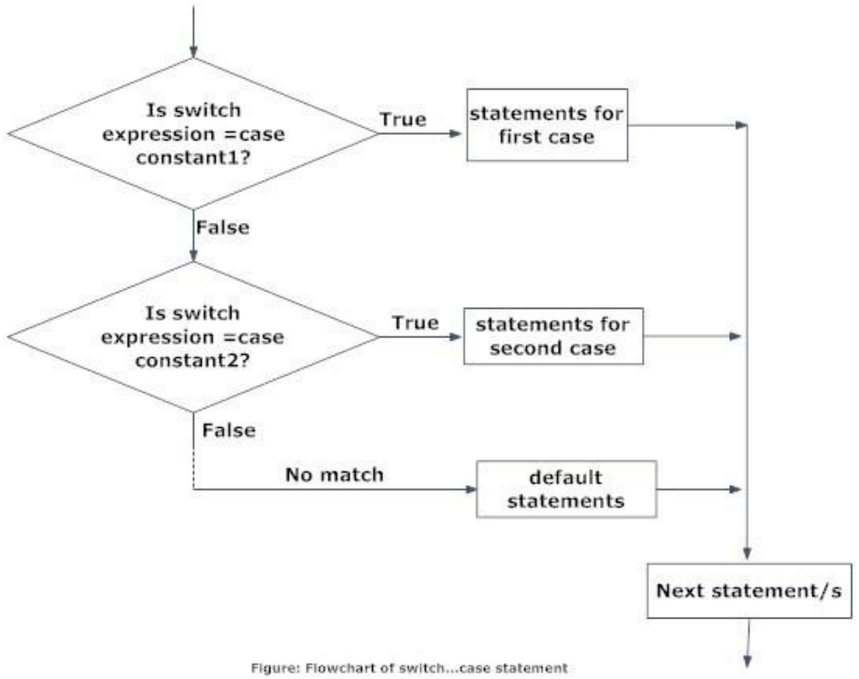

Is switch expression =case constant1?
True
statements for first case
False
Is switch expression =case constant2?
True
statements for second case
False
No match
default statements
Next statement/s

Figure: Flowchart of switch...case statement

The goto Statement in C++

What is goto?

The goto statement allows the program to jump to a labeled statement.

It provides unconditional control transfer.

Use it carefully – excessive use can make the code hard to read and maintain (often called "spaghetti code").

Example :

```
#include <iostream>
using namespace std;

int main() {
   int number = 1;

   start:
   cout << number << " ";
   number++;

   if (number <= 5)
      goto start;

   return 0;
}
```

Exercise 1: Write a C++ program that uses a loop to find the first multiple of 5 between 1 and 50. Use the break statement to exit the loop once the multiple is found.

```
#include <iostream> using namespace std; int main()
{
  for (int i = 1; i <= 50; ++i) { if (i % 5 == 0) {
      cout << "The first multiple of 5 between 1 and 50 is: " << i << endl; break;
    }
  }
}
```

Exercise 2: Write a C++ program that uses a loop to print all numbers from 1 to 10, except the number 4. Use the continue statement to skip the number 4.

```
#include <iostream> using namespace std; int main() {
  for (int i = 1; i <= 10; ++i) { if (i == 4) {
      continue;
    }
    cout << i << " ";
  }
  cout << endl;
}
```

Exercise 3: Write a C++ program that reads an integer from the keyboard and prints the corresponding day of the week. Use a switch statement to handle the different cases (1 for Monday, 2 for Tuesday, etc.).

```
#include <iostream>
using namespace std; int main() {
 int day;
 cout << "Enter a number (1-7) to get the corresponding day of the week: "; cin >> day;
 switch (day) { case 1:
     cout << "Monday" << endl; break;
   case 2:
     cout << "Tuesday" << endl; break;
   case 3:
     cout << "Wednesday" << endl; break;
     case 4:
     cout << "Thursday" << endl; break;
     case 5:
     cout << "Friday" << endl; break;
     case 6:
     cout << "Saturday" << endl; break;
     case 7:
     cout << "Sunday" << endl; break;
     default:
     cout << "Invalid input! Please enter a number between 1
     and 7." << endl;
     }
     }
```

Exercise 4: Write a C++ program that reads a student's score (0-100) from the keyboard and determines their grade based on the score. Use a switch statement to handle the different

grade ranges (A, B, C, D, F).

```
#include <iostream>
using namespace std; int main() {
  int score; char grade;
  cout << "Enter the student's score (0-100): "; cin >> score;
  switch (score / 10) { case 10:
  case 9: grade = 'A'; break;
  case 8:
  grade = 'B'; break;
  case 7:
  grade = 'C'; break;
  case 6: grade = 'D'; break;
  default:
  grade = 'F'; break;
  }
  cout << "The grade is: " << grade << endl;
  }
```

Home Works

- **Task:** Write a C++ program that reads student scores from the keyboard until a negative score is entered. Use a switch statement to determine the grade based on the score and display the result. Use continue to skip invalid scores (scores not in the range 0-100).
- **Requirements**:

1. Continuously read scores from the user until a negative score is entered.
2. Use continue to skip invalid scores (less than 0 or greater than 100).
3. Use a switch statement to assign a grade (A, B, C, D, F) based on the score range.
4. Use break to exit the loop when a negative score is entered.

THANK
YOU