Loops and Decisions

Relational operators

- A **relational operator** is a programming language construct or operator that tests or defines some kind of relation between two entities. These include numerical equality (e.g., 5 = 5) and inequalities (e.g., 4 ≥ 3).
- The relational operators are often used to create a test expression that controls program flow.
- This type of expression is also known as a Boolean expression because they create a Boolean answer or value when evaluated.
- **Relational operators:**

  < , <= , > , >= , == , !=

- Example:

```
int main()
{
    int no;
    cout<<“enter a no”;
    cin>>no;
    cout<<“no<10 is”<<(no<10);
     cout<<“no>10 is”<<(no>10);
     cout<<“no==10 is”<<(no==10);
}
```

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.
- Programming languages provide various control structures that allow for more complicated execution paths.

Types of Loops

- for Loop:
  - A for loop is a repetition control structure which allows us to write a loop that is executed a specific number of times.

Syntax:

```
for (initialization expr; test expr; update expr)
{
 // body of the loop
// statements we want to execute
}
```

Example

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

int main()
{
    int i=0;

    for (i = 1; i <= 10; i++)
    {
        cout<< "Hello World”;
    }

    return 0;
}
```

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

int main()
{
    int i = 0;

    for (i = 1; i <= 10; i += 2) {
        cout << i << "\n";
    }

    return 0;
}
```

- While loop:
- While studying **for loop** we have seen that the number of iterations is known beforehand.
- while loops are used in situations where we do not know the exact number of iterations of loop beforehand.
- Syntax:

```
initialization expression;
while (test_expression)
{
// statements
update_expression;
}
```

Example

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

int main()
{
    // initialization expression
    int i = 1;

    // test expression
    while (i < 6)
    {
        cout<< "Hello World\n”;

        // update expression
        i++;
    }

    return 0;
}
```

- do while loop
- In do while loops also the loop execution is terminated on the basis of test condition.
- The main difference between do while loop and while loop is in do while loop the condition is tested at the end of loop body.
- In do while loop the loop body will execute at least once irrespective of test condition.

- **Syntax:**
- **initialization expression;**
- do
- {
- // statements
- **update_expression;**
- } while (**test_expression**);

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

int main()
{
    int i = 2;                  // Initialization expression

    do
    {
        // loop body
        cout<< "Hello World\n”;

        // update expression
        i++;

    }  while (i < 1);                  // test expression

    return 0;
}
```

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

int main()
{
    // initialization expression
    int i = 1;

    // test expression
    while (i > -5) {
        cout << i << "\n";

        // update expression
        i--;
    }
   return 0;
}
```

- Nested Loops:
  - **Nested loop** means a loop statement inside another loop statement. That is why nested loops are also called as "**loop inside loop**".
  - Syntax for Nested For loop:

```
for ( initialization; condition; increment )
{
      for ( initialization; condition; increment )
      {
            // statement of inside loop
       }
      // statement of outer loop
 }
```

- Syntax for Nested While loop:

```
while(condition)
 {
     while(condition)
      {
         // statement of inside loop
      }
     // statement of outer loop
 }
```

- Syntax for Nested Do-While loop:

```
do
{
      do
     {
          // statement of inside loop

     } while(condition);

     // statement of outer loop

 } while(condition);
```

- There is no rule that a loop must be nested inside its own type. In fact, there can be any type of loop nested inside any type and to any level.
- Syntax:

```
do
{
      while(condition)
      {
           for ( initialization; condition; increment )
           {    // statement of inside for loop
           }
            // statement of inside while loop
       }
      // statement of outer do-while loop
 }   while(condition);
```

- break statement:
  - When the **break** statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop.
  - It can be used to terminate a case in the **switch** statement.
  - If you are using nested loops (i.e., one loop inside another loop), the break statement will stop the execution of the innermost loop and start executing the next line of code after the block.

Example

```
cout << "Enter student names (max of 50).  Enter q to quit" << endl;

int count = 0;

while (count ++ < 50)
{
        cout << "Name: ";
        cin >> inputString;
        if (inputString == "q")
                break;
}
```

Contd..

```
string inputString;

cout << "Enter student names (max of 50).  Enter q to quit" << endl;

int count = 0;
bool continueInput  = true;

while (count ++ < 50 && continueInput)
{
        cout << "Name: ";
        cin >> inputString;
        if (inputString == "q")
                continueInput = false;
}
```

Check to see if user wants to continue

Change continueInput to false so looping will stop

```
#include <iostream>
using namespace std;
int main ()
 {
      int a = 10;
      do
     {
           cout << "value of a: " << a << endl;
           a = a + 1;
           if( a > 15)
           {
                break;
           }
      } while( a < 20 );
     return 0;
 }
```

- Output:

  value of a:10

  value of a:11

  value of a:12

  value of a:13

  value of a:14

  value of a:15

- continue statement:
  - The **continue** statement works somewhat like the break statement. Instead of forcing termination, however, continue forces the next iteration of the loop to take place, skipping any code in between.
  - For the **for** loop, continue causes the conditional test and increment portions of the loop to execute. For the **while** and **do...while** loops, program control passes to the conditional tests.

Example

```
#include <iostream>
using namespace std;
 int main ()
{
      int a = 10;
      do
       {
            if( a == 15)
            {   a = a + 1;
                continue;
            }
            cout << "value of a: " << a << endl;
            a = a + 1;
       } while( a < 20 );
      return 0;
 }
```

Output:

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 16

value of a: 17

value of a: 18

value of a: 19

- goto statement:
  - A **goto** statement provides an unconditional jump from the goto to a labeled statement in the same function.
  - Use of **goto** statement is highly discouraged because it makes difficult to trace the control flow of a program

Example

```
#include <iostream>
using namespace std;
void func(int num)
{
    if (num % 2 == 0)
        goto even;
    else
        goto odd;

  even:
     cout << num << " is even";
     return;
  odd:
     cout << num << " is odd";
}
 int main()
{
    int num = 26;
    fun(num);
    return 0;
}
```

Example

```
int main()
{
  int i, j, k;
  for(i=0;i<10;i++)
  {
    for(j=0;j<5;j++)
    {
      for(k=0;k<3;k++)
      {
        printf("%d %d %d\n",i,j,k);
        if(j == 3)
        {
          goto out;
        }
      }
    }
  }
  out:
  printf("came out of the loop");
}
```

Find Output

```
#include <iostream>
using namespace std;
int main()
{
    int i = 3;
    while (i--)
    {
        int i = 100;
        i--;
        cout<< i;
    }
    return 0;
}
```

Find Output

```
#include<iostream>
int main()
{
    int i = -5;
    while (i <= 5)
    {
        if (i >= 0)
            break;
        else
        {
            i++;
            continue;
        }
        cout<<“Hello World”;
    }
    return 0;
}
```

Decision Control

- if
- if else
- conditional operator

- if statement:
  - It is used to decide whether a certain statement or block of statements will be executed or not.

Syntax:

```
if(condition)
{
    // Statements to execute if
    // condition is true
}
```

- if else statement:
  - We can use the *else* statement with *if* statement to execute a block of code when the condition is false.
  - Syntax:

```
if (condition)
{
 // Executes this block if condition is true
}
else
{
     // Executes this block if  condition is false
}
```

- Nested if else statement:
  - Nested if else statement means one if else statement inside another if else statement.
  - Syntax:

```
if(condition)
{
}
else
{
    if(condition)
    {
    }
    else
    {
    }
}
```

```
if(condition)
{
    if(condition)
    {
    }
     else
     {
     }
 }
else
 {
 }
```

Example

```
int main()
{
    int a,b,c;
    cout<<“enter nos”;
    cin>>a>>b>>c;
    if(a>b && a>c)
        cout<<a;
    if(b>a && b>c)
        cout<<b;
    if(c>a && c>b)
        cout<<c;
}
```

Example

```
int main()
{
    int a,b,c;
    cout<<“enter nos”;
    cin>>a>>b>>c;
    if(a>=b && a>=c)
        cout<<a;
    if(b>=a && b>=c)
        cout<<b;
    if(c>=a && c>=b)
        cout<<c;
}
```

Example

```
int main()
{
    int a,b,c;
    cout<<“enter nos”;
    cin>>a>>b>>c;
    if(a>=b && a>=c)
        cout<<a;
    if(b>=a && b>=c)
        cout<<b;
    else
        cout<<c;
}
```

Example

```
int main()
{
    int a,b,c;
    cout<<“enter nos”;
    cin>>a>>b>>c;
    if(a>=b && a>=c)
        cout<<a;
    else
    {
        if(b>=a && b>=c)
            cout<<b;
        else
            cout<<c;
    }
```

Example

```
int main()
{
    int a,b,c;
    cout<<“enter nos”;
    cin>>a>>b>>c;
    if(a>=b && a>=c)
        cout<<a;
    else
    {
        if( b>=c)
            cout<<b;
        else
            cout<<c;
    }
```

Example

```
int main()
{
    int a,b,c;
    cout<<“enter nos”;
    cin>>a>>b>>c;
    if(a>b)
    {
        if(a>c)
            cout<<a;
        else
            cout<<c;
    }
    else
    {
        if( b>c)
            cout<<b;
        else
            cout<<c;
    }
```

- if else ladder:
- Syntax:

```
if (condition)
    statement;
else if (condition)
    statement;
else if (condition)
    statement;
.
.
else
    statement;
```

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

int main()
{
    int i = 20;

    if (i == 10)
        cout<<"i is 10";
    else if (i == 15)
        cout<<"i is 15";
    else if (i == 20)
        cout<<"i is 20";
    else
        cout<<"i is not present";
}
```

- Matching the else:
- #include <iostream>
- using namespace std;
- int main()
- {
- int a, b, c;
- cout << “Enter three numbers, a, b, and c:\n”;
- cin >> a >> b >> c;
- if(a==b)
- if(b==c)
- cout << “a, b, and c are the same\n”;
- else
- cout << “a and b are different\n”;
- return 0;
- }

- Output:(a=2,b=3,c=3)
- Nothing is printed

- Matching the else:
- #include <iostream>
- using namespace std;
- int main()
- {
- int a, b, c;
- cout << “Enter three numbers, a, b, and c:\n”;
- cin >> a >> b >> c;
- if(a==b)
- if(b==c)
- cout << “a, b, and c are the same\n”;
- else
- cout << “b and c are different\n”;
- return 0;
- }

- Output:(a=2,b=2,c=3)
- b & c are different

getche() function

- this is a non-standard function present in conio.h
- It reads a single character from the keyboard and displays immediately on output screen without waiting for enter key.
- Syntax:

```
int getche(void);
```

```
include<iostream>
using namespace std;
#include <conio.h>            //for getche()
int main()
{
int chcount=0;
int wdcount=1;
char ch = ‘a’;
cout << “Enter a phrase: “;
while( ch != ‘\r’ )
{
ch = getche();
if( ch==’ ‘ )
wdcount++;
else //otherwise,
chcount++;
}
cout << “\nWords=” << wdcount << endl
<< “Letters=” << (chcount-1) << endl;
return 0;
}
```

- Output:

  for while and do

  words=4

  letters=13

Find output

```
#include <iostream>
using namespace std;
int main ()
{
    int n;
    for (n = 5; n > 0; n--)
    {
        cout << n;
        if (n == 3)
            break;
    }
    return 0;
}
```

- Output:

  543

Find output

```
#include <iostream>
using namespace std;
int main()
{
    int a = 10;
    if (a < 15)
    {
        time:
        cout << a;
        goto time;
    }
    break;
    return 0;
}
```

- Output:

  compile time error

Find output

```
#include <iostream>
using namespace std;
int main()
{
    int n = 15;
    for ( ; ; )
      cout << n;
    return 0;
}
```

- Output:

  infinite times of printing n

Conditional Operator

- The conditional operator is kind of similar to the if-else statement as it does follow the same algorithm as of if-else statement.
- Syntax:

  variable = Expression1 ? Expression2 : Expression3

- It can be visualized into if-else statement as:

```
if(Expression1)
 {   variable = Expression2; }
else
{ variable = Expression3; }
```

```
bool y;
if (x == 42){
  y = true;
}
else {
  y = false;
}
```

code in one line:

```
bool y; if (x == 42) {y = true;} else {y = false;}   // messy
```

using conditional operator:

```
bool y = (x == 42) ? true : false;
```

- When to avoid it?

```
if (x == 42) {
  std::cout << "code is executed" << std::endl;
  code();
}
else {
  std::cout << "nothing happened" << std::endl;
}
```

vs

```
(x == 42) ? ((std::cout << "code is executed" << std::endl),
   code())
: (void)(std::cout << "nothing happened" << std::endl);
```

Switch statement

- Switch case statements are a substitute for long if statements that compare a variable to several integral values
  - The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression.
  - Switch is a control statement that allows a value to change control of execution.

- Syntax:

```
switch(expression)
{
    case constant : code;
    case constant : code;
    case constant : code;
    default: code;
}
```

- Syntax:

```
switch(expression)
{
    case constant : code; break;
    case constant : code; break;
    case constant : code; break;
    default: code;
}
```

```
include <iostream>
using namespace std;

int main() {
int x = 2;
    switch (x)
    {
        case 1:
            cout << "Choice is 1";
            break;
        case 2:
            cout << "Choice is 2";
            break;
        case 3:
            cout << "Choice is 3";
            break;
        default:
            cout << "Choice other than 1, 2 and 3";
            break;
    }
return 0;
}
```

Find output

```
#include <iostream>
using namespace std;
int main()
{
    int num = 2;
    switch (num + 2)
    {
    case 1:
        cout<<"Case 1: “;
    case 2:
        cout<<"Case 2: “;
    case 3:
        cout<<"Case 3: “;
    default:
        cout<<"Default: “;
    }
    return 0;
}
```

Find output

```
#include<iostream>
using namespace std;
int main()
{
    switch(2)
    {
    case 1:
        cout<<"No“;
    case 2:
        cout<<“hello";
        goto Label;
    case 3:
        cout<<“yes";
    case 4:Label:
        cout<<“world";
    }
    return 0;
}
```

Find output

```
#include<iostream>
using namespace std;
int main()
{
    int colour = 2;
    switch (colour) {
    case 0:
        cout<<"Black”;
    case 1:
        cout<<"Red”;
    case 2:
        cout<<"Aqua”;
    case 3:
        cout<<"Green";
    default:
        cout<<"Other”;
    }
}
```

Find output

```
#include <iostream>
using namespace std;
int main()
{
    int i = 0;
    switch (i)
    {
        case '0': cout<<“hello”;
                break;
        case '1': cout<<“world";
                break;
        default: cout<<“helloworld”;
    }
    return 0;
}
```

Find output

```
# include <iostream>
using namespace std;
int main()
{
   int i = 0;
   for (i=0; i<20; i++)
   {
     switch(i)
     {
       case 0:   i += 5;
       case 1:   i += 2;
       case 5:   i += 5;
       default:   i += 4;   break;
     }
     cout<< i;
   }
   return 0;
}
```

Find output

```
#include <iostream>
using namespace std;
int main()
{
    int x = 3;
    if (x == 2); x = 0;
    if (x == 3) x++;
    else x += 2;

    cout<<"x = “<< x;

    return 0;
}
```

Find output

```
#include<iostream>
using namespace std;
int main()
{
    int a = 5;
    switch(a)
    {
    default: a = 4;
    case 6: a--;
    case 5: a = a+1;
    case 1: a = a-1;
    }
    cout<< a;
    return 0;
}
```

Logical Operators

- They are used to combine two or more conditions/constraints or to complement the evaluation of the original condition under consideration.
- Logical AND operator:
  - The **‘&&’** operator returns true when both the conditions under consideration are satisfied. Otherwise it returns false. For example, **a && b** returns true when both a and b are true (i.e. non-zero).

- Logical OR operator:
  - The '**||**' operator returns true even if one (or both) of the conditions under consideration is satisfied. Otherwise it returns false.
  - For example, **a || b** returns true if one of a or b or both are true (i.e. non-zero). Of course, it returns true when both a and b are true.
- Logical NOT operator:
  - The '**!**' operator returns true the condition in consideration is not satisfied. Otherwise it returns false. For example, **!a** returns true if a is false, i.e. when a=0.

```
#include <iostream>
using namespace std;
int main()
{
    int a = 10, b = 4, c = 10, d = 20;

    if (a > b && c == d)
        cout << "a is greater than b AND c is equal to d\n";
    else
        cout << "AND condition not satisfied\n";

    if (a > b || c == d)
        cout << "a is greater than b OR c is equal to d\n";
    else
        cout << "Neither a is greater than b nor c is equal "
             " to d\n";
```

```
if (!a)
        cout << "a is zero\n";
    else
        cout << "a is not zero";

    return 0;
}
```

Output:

AND condition not satisfied

a is greater than b OR c is equal to d

a is not zero

Find Output

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

int main()
{
    int a = 1;
    int b = 1;
    int c = a  ||  --b;
    int d = a--  &&  --b;
    cout<< a<< b<< c<< d;
    return 0;
}
```

- Output:

  a = 0, b = 0, c = 1, d = 0