Decision making in C

Decision making is about deciding the order of execution of statements based on certain conditions or repeat a group of statements until certain specified conditions are met. C language handles decision-making by supporting the following statements,

- `if` statement
- `switch` statement
- conditional operator statement (`? :` operator)
- `goto` statement

Decision making with `if` statement

The `if` statement may be implemented in different forms depending on the complexity of conditions to be tested. The different forms are,

1. Simple `if` statement
2. `if....else` statement
3. Nested `if....else` statement
4. Using `else if` statement

Simple `if` statement

The general form of a simple `if` statement is,

```
if(expression)

{

    statement inside;

}

    statement outside;
```

If the *expression* returns true, then the **statement-inside** will be executed, otherwise **statement-inside** is skipped and only the **statement-outside** is executed.

Example:

```
#include <stdio.h>

void main( )
```

```
{
    int x, y;
    x = 15;
    y = 13;
    if (x > y )
    {
        printf("x is greater than y");
    }
}
```

```
x is greater than y
```

`if...else` statement

The general form of a simple `if...else` statement is,

```
if(expression)
{
    statement block1;
}
else
{
    statement block2;
}
```

If the *expression* is true, the **statement-block1** is executed, else **statement-block1** is skipped and **statement-block2** is executed.

Example:

```
#include <stdio.h>
void main( )
{
    int x, y;
    x = 15;
    y = 18;
    if (x > y )
```

```
    {
        printf("x is greater than y");
    }
    else
    {
        printf("y is greater than x");
    }
}
```

```
y is greater than x
```

Nested `if....else` statement

The general form of a nested `if...else` statement is,

```
if( expression )
{
    if( expression1 )
    {
        statement block1;
    }
    else
    {
        statement block2;
    }
}
else
{
    statement block3;
}
```

if *expression* is false then **statement-block3** will be executed, otherwise the execution continues and enters inside the first `if` to perform the check for the

next `if` block, where if *expression 1* is true the **statement-block1** is executed otherwise **statement-block2** is executed.

Example:

```
#include <stdio.h>

void main( )

{

    int a, b, c;

    printf("Enter 3 numbers...");

    scanf("%d%d%d",&a, &b, &c);

    if(a > b)

    {

        if(a > c)

        {

            printf("a is the greatest");

        }

        else

        {

            printf("c is the greatest");

        }

    }

    else

    {

        if(b > c)

        {

            printf("b is the greatest");

        }

        else

        {

            printf("c is the greatest");

        }

    }
```

```
}
```

`else if` ladder

The general form of else-if ladder is,

```
if(expression1)
{
    statement block1;
}
else if(expression2)
{
    statement block2;
}
else if(expression3 )
{
    statement block3;
}
else
    default statement;
```

The expression is tested from the top(of the ladder) downwards. As soon as a **true** condition is found, the statement associated with it is executed.

Example :

```
#include <stdio.h>
void main( )
{
    int a;
    printf("Enter a number...");
    scanf("%d", &a);
    if(a%5 == 0 && a%8 == 0)
    {
        printf("Divisible by both 5 and 8");
    }
    else if(a%8 == 0)
```

```
    {
        printf("Divisible by 8");
    }
    else if(a%5 == 0)
    {
        printf("Divisible by 5");
    }
    else
    {
        printf("Divisible by none");
    }
}
```

Points to Remember

1. In `if` statement, a single statement can be included without enclosing it into curly braces `{ ... }`

```
2. int a = 5;
3. if(a > 4)
       printf("success");
```

   No curly braces are required in the above case, but if we have more than one statement inside `if` condition, then we must enclose them inside curly braces.

4. `==` must be used for comparison in the expression of `if` condition, if you use `=` the expression will always return **true**, because it performs assignment not comparison.
5. Other than **0(zero)**, all other values are considered as **true**.

```
6. if(27)
       printf("hello");
```

   In above example, **hello** will be printed.