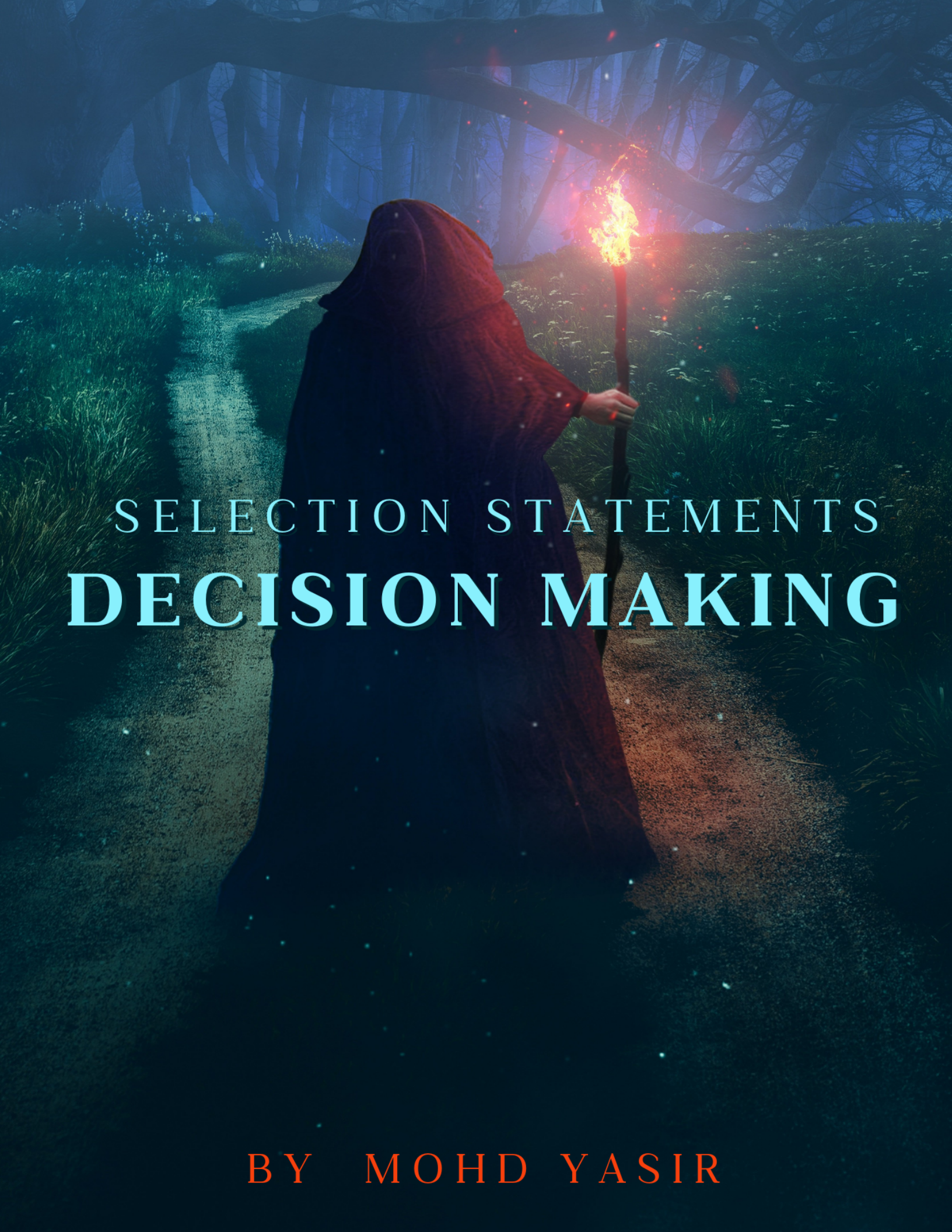

SELECTION STATEMENTS
DECISION MAKING
BY MOHD YASIR

# UNIT - II

## SELECTION STATEMENTS(DECISION MAKING):

### IF AND SWITCH STATEMENTS :

We have a number of situations where we may have to change the order of execution of statements based on certain conditions or repeat a group of statements until certain specified conditions are met.

The if statement is a two way decision statement and is used in conjunction with an expression. It takes the following form

If(test expression)

If the test expression is true then the statement block after if is executed otherwise it is not executed

```
if (test expression)
{
    statement block;

}
statement–x ;
```

only statement–x is executed.

```
/* program for if  */
#include<stdio.h>
main()
{
    int a,b;
    printf("Enter two numbers");
```

```
        scanf("%d%d",&a,&b):
        if a>b
                printf(" a is greater");
        if b>a
                printf("b is greater");
}
```

**The if –else statement:**

If your have another set of statement to be executed if condition is false then if-else is used

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

```
/* program for if-else  */
#include<stdio.h>
main()
{
        int a,b;
```

```
        printf(“Enter two numbers”);
        scanf(“%d%d”,&a,&b):
        if a>b
                printf(“ a is greater”)
        else
                printf(“b is greater”);
}
```

**Nesting of if..else statement :**

If more than one if else statement

```
if(text cond1)
{
        if (test expression2
        {
                statement block1;
        }
        else
        {
                statement block 2;
        }
}
else
{
        statement block2;
```

}

statement-x ;

if else ladder

```
if(condition1)
        statement1;
else if(condition2)
        statement 2;
else if(condition3)
        statement n;
else
        default statement.
statement-x;
```

The nesting of if-else depends upon the conditions with which we have to deal.

## THE SWITCH STATEMENT:

If for suppose we have more than one valid choices to choose from then we can use switch statement in place of if statements.

```
switch(expression)
        {.
                case value-1
```

```
                    block-1
                    break;
        case value-2
                    block-2
                    break;
            --------
            --------
        default:
                    default block;
                    break;
    }
    statement–x
```

In case of

```
    if(cond1)
        {
        statement-1
        }
    if (cond2)
        {
            statement 2
        }
```

```
/*   program to implement switch   */
#include<stdio.h>
main()
{
        int marks,index;
        char grade[10];
        printf("Enter your marks");
        scanf("%d",&marks);
        index=marks/10;
        switch(index)
        {
                case 10 :
                case 9:
                case 8:
                case 7:
                case 6: grade="first";
                        break;
                case 5 : grade="second";
                        break;
                case 4 : grade="third";
                        break;
                default : grade ="fail";
                        break;
        }
```

```
    printf("%s",grade);
}
```

## LOOPING :

Some times we require a set of statements to be executed repeatedly until a condition is met.

We have two types of looping structures. One in which condition is tested before entering the statement block called entry control.

The other in which condition is checked at exit called exit controlled loop.

## WHILE STATEMENT :

```
    While(test condition)
    {
        body of the loop
    }
```

It is an entry controlled loop. The condition is evaluated and if it is true then body of loop is executed. After execution of body the condition is once again evaluated and if is true body is executed once again. This goes on until test condition becomes false.

```
/* program for while   */
#include<stdio.h>
main()
{
    int count,n;
    float x,y;
    printf("Enter the values of x and n");
```

```
        scanf("%f%d",&x,&n);
        y=1.0;
        count=1;
        while(count<=n)
        {
                y=y*x;
                count++;
        }
        printf("x=%f; n=%d; x to power n = %f",x,n,y);
}
```

## DO WHILE STATEMENT :

The while loop does not allow body to be executed if test condition is false. The do while is an exit controlled loop and its body is executed at least once.

```
        do
        {
                body
        }while(test condition);
```

```
/* printing multiplication table  */
#include<stdio.h>
#define COL 10
#define ROW 12
main()
```

```
{
        int row,col,y;
        row=1;
        do
        {
                col=1;
                do
                {
                        y=row*col;
                        printf("%d",y);
                        col=col+1;
                }while(col<=COL);
                printf("\n");
                row=row+1;
        }while(row<=ROW);
}
```

## THE FOR LOOP :

It is also an entry control loop that provides a more concise structure

```
for(initialization; test control; increment)
{
        body of loop
}
```

```
/*  program of for loop */

#include<stdio.h>
main()

{

        long int p;

        int n;

        double q;

        printf("2 to power n   ");

        p=1;

        for(n=0;n<21;++n)

        {

                if(n==0)

                        p=1;

                else

                        p=p*2;

                q=1.0/(double)p;

                printf("%101d%10d",p,n);

        }

}
```

## UNCONDITIONAL STATEMENTS:

## BREAK STATEMENT:

This is a simple statement. It only makes sense if it occurs in the body of a `switch`, `do`, `while` or `for` statement. When it is executed the control of flow jumps to the statement immediately following the body of the statement containing the `break`. Its use is widespread in `switch` statements, where it is more or less essential to get the control .

The use of the `break` within loops is of dubious legitimacy. It has its moments, but is really only justifiable when exceptional circumstances have happened and the loop has to be abandoned. It would be nice if more than one loop could be abandoned with a single break but that isn't how it works. Here is an example.

```
#include <stdio.h>
#include <stdlib.h>
main(){
    int i;

    for(i = 0; i < 10000; i++){
        if(getchar() == 's')
            break;
        printf("%d\n", i);
    }
    exit(EXIT_SUCCESS);
}
```

It reads a single character from the program's input before printing the next in a sequence of numbers. If an '`s`' is typed, the break causes an exit from the loop.

If you want to exit from more than one level of loop, the `break` is the wrong thing to use.

## CONTINUE STATEMENT:

This statement has only a limited number of uses. The rules for its use are the same as for `break`, with the exception that it doesn't apply to `switch` statements. Executing a `continue` starts the next iteration of the smallest enclosing `do`, `while` or `for` statement immediately. The use of `continue` is largely restricted to the top of loops, where a decision has to be made whether or not to execute the rest of the body of the loop. In this example it ensures that division by zero (which gives undefined behaviour) doesn't happen.

```
#include <stdio.h>
#include <stdlib.h>
main(){
    int i;

    for(i = -10; i < 10; i++){
        if(i == 0)
            continue;
        printf("%f\n", 15.0/i);
        /*
         * Lots of other statements .....
         */
    }
    exit(EXIT_SUCCESS);
```

```
}
```

The `continue` can be used in other parts of a loop, too, where it may occasionally help to simplify the logic of the code and improve readability. `continue` has no special meaning to a `switch` statement, where `break` does have. Inside a `switch`, `continue` is only valid if there is a loop that encloses the `switch`, in which case the next iteration of the loop will be started.

There is an important difference between loops written with `while` and `for`. In a `while`, a continue will go immediately to the test of the controlling expression. The same thing in a `for` will do two things: first the update expression is evaluated, then the controlling expression is evaluated.

## GOTO AND LABELS:

In C, it is used to escape from multiple nested loops, or to go to an error handling exit at the end of a function. You will need a *label* when you use a `goto`; this example shows both.

```
goto L1;
/* whatever you like here */
L1: /* anything else */
```

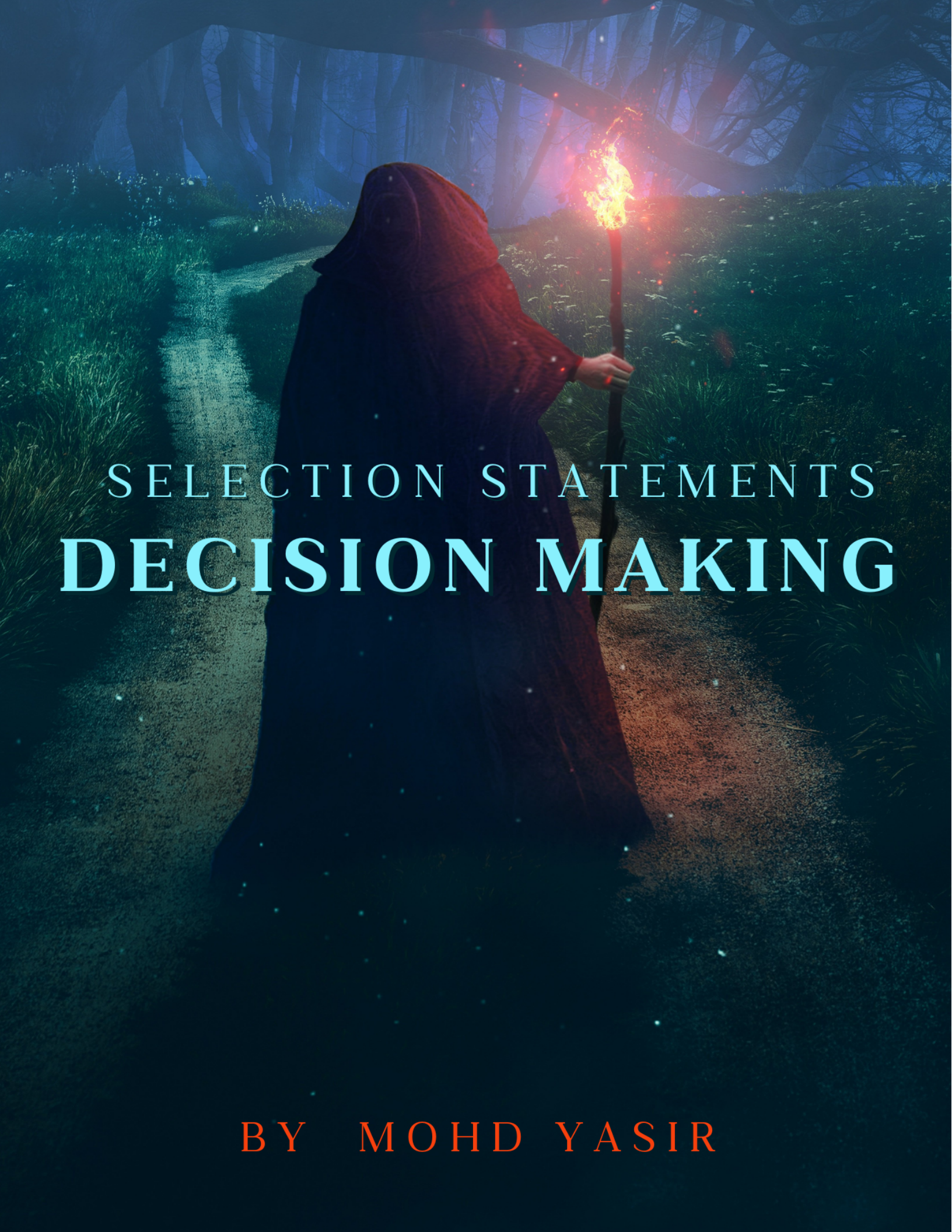
A label is an identifier followed by a colon. Labels have their own ‘name space’ so they can't clash with the names of variables or functions. The name space only exists for the function containing the label, so label names can be re-used in different functions. The label can be used before it is declared, too, simply by mentioning it in a `goto` statement.

Labels must be part of a full statement, even if it's an empty one. This usually only matters when you're trying to put a label at the end of a compound statement—like this.

```
label_at_end: ; /* empty statement */
}
```

The `goto` works in an obvious way, jumping to the labelled statements. Because the name of the label is only visible inside its own function, you can't jump from one function to another one.

It's hard to give rigid rules about the use of gotos but, as with the do, continue and the break (except in switch statements), over-use should be avoided. More than one goto every 3–5 functions is a symptom that should be viewed with deep suspicion.


SELECTION STATEMENTS
DECISION MAKING
BY MOHD YASIR