Sequence Control and Subprograms

By

Prof Muhammad Iqbal Bhat

Government Degree College Beerwah

Topics

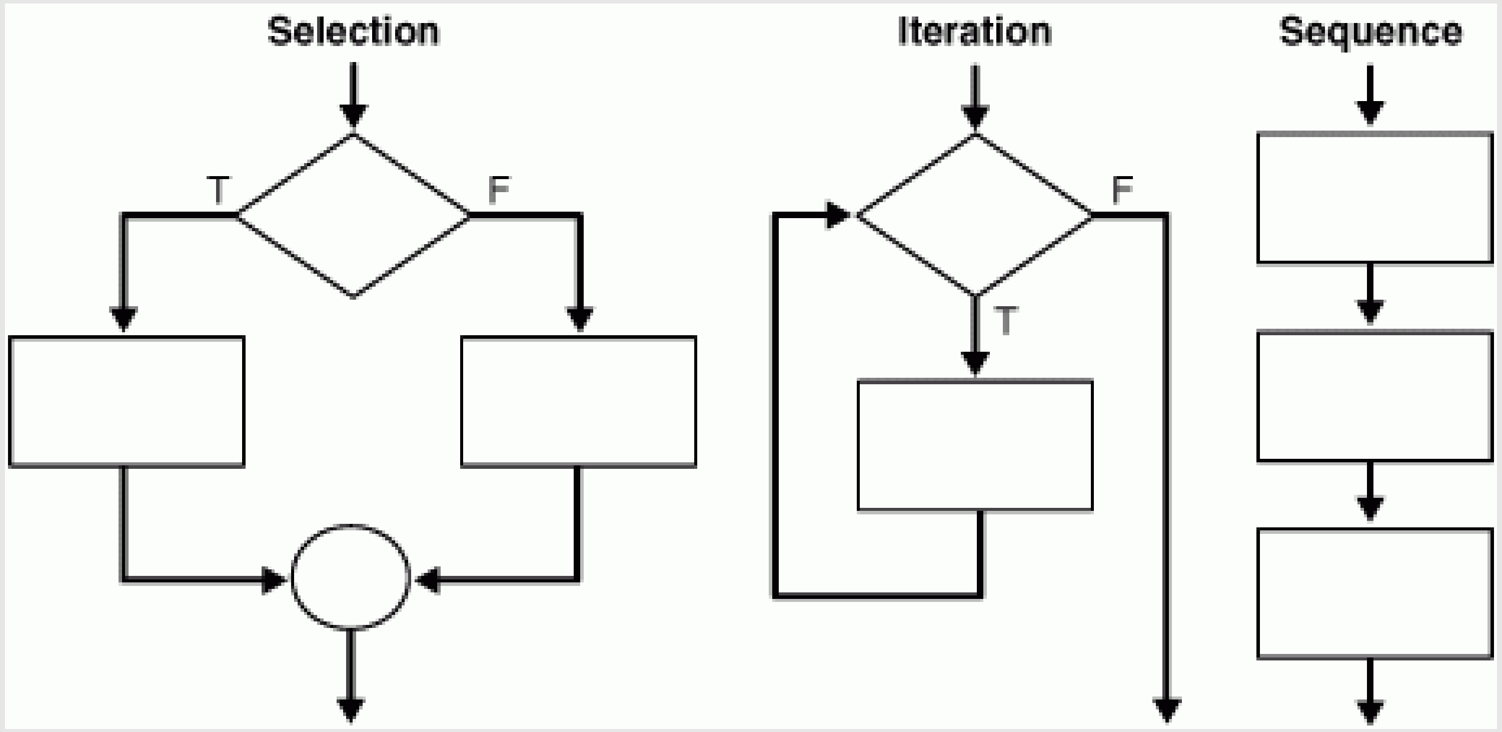
- Sequence Control
- Implicit and Explicit Sequence Control
- Subprogram Control

Introduction:

Programming languages provide ways to control the order of execution of statements.

Sequence control determines the order in which statements are executed.

Subprogram control refers to how a program can call and return from subprograms or functions.



Sequence Control

Programming languages provide ways to control the order of execution of statements.

This is important because it allows us to write programs that solve complex problems by breaking them down into smaller, more manageable pieces.

Sequence control is the mechanism by which a programming language controls the order in which statements are executed in a program.

Types of Sequence Control:

- Implicit Sequence Control
- Explicit Sequence Control

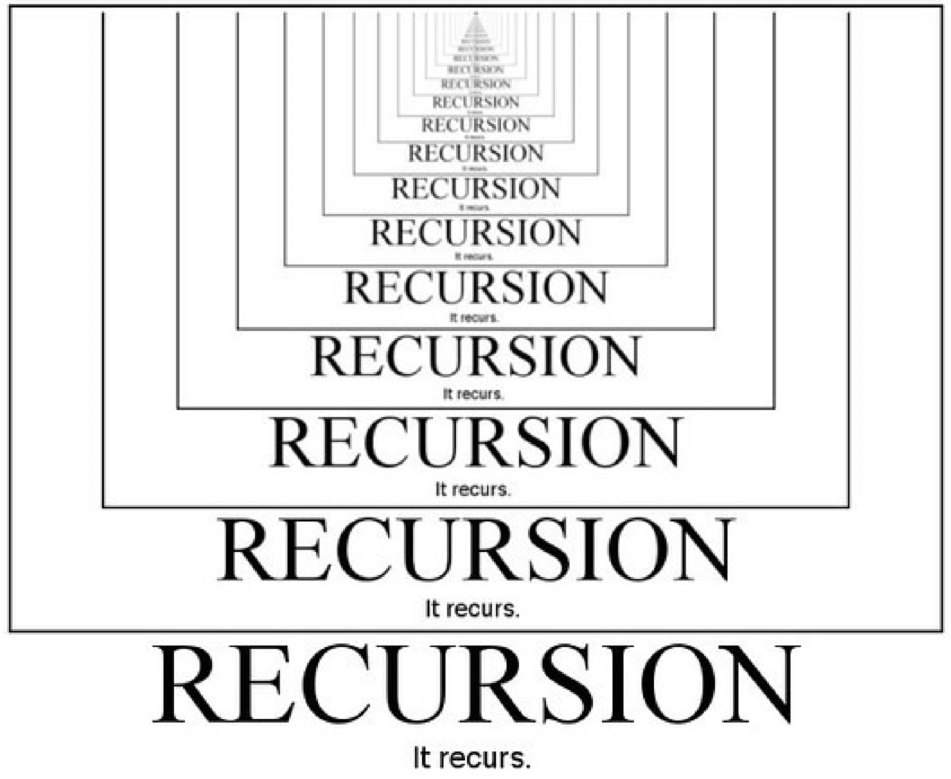
Types of Sequence Control:

Implicit Sequence Control:

- Implicit sequence control is the default behavior of a programming language, where statements are executed in the order in which they appear in the program.
- Implicit sequence control relies on the order in which statements appear in the program.
- It is the default sequence and defined by the Language
- Examples: Expressions, Statements etc

Explicit Sequence Control:

- Explicit sequence control, on the other hand, is where the programmer explicitly specifies the order in which statements are executed using control structures.
- Implicit sequence control relies on the order set by the programmer in form of conditional and looping statements in the program.
- The most common types of control structures used for explicit sequence control are loops and conditionals.



Subprograms

Simple vs Recursive

Subprogram Control:

Subprograms are self-contained units of code that perform specific tasks and can be called by other parts of a program. They are commonly used in programming languages to break down complex problems into smaller, more manageable pieces.

Subprogram control refers to how a program can call and return from subprograms or functions.

Simple subprogram control involves calling a subprogram and returning to the calling program when the subprogram is finished. The subprogram is executed and then control is returned to the calling program.

In programming languages, simple subprogram control is typically accomplished using a call statement and a return statement. The call statement invokes the subprogram, and the return statement returns control to the calling program.

Understanding subprogram control is essential for developing effective programs that are modular, easy to understand, and easy to maintain.

Types of Subprograms:

- Simple Subprogram
- Recursive Subprogram

Simple Subprogram:

Simple subprogram control involves calling a subprogram and returning to the calling program when the subprogram is finished.

In other words, the subprogram is executed and then control is returned to the calling program.

Simple subprogram control is useful for breaking down complex problems into smaller subproblems that can be solved using the same algorithm. By encapsulating these subproblems in subprograms, we can write more efficient, easier to understand, and more maintainable code.

The code for the subprogram can be written once and then called multiple times from different parts of the program.

Simple Subprogram

```
 return_type function_name(parameter_type parameter_name)
 {
    // Code to perform the task
    return result;
 }

float area_of_circle(float radius) {
   const float pi = 3.14159;
   float area = pi * radius * radius;
   return area;
}

Int main(){
              /* Call subprogram
              result = area_of_circle(10.1);
}
```

Recursive Subprogram:

Recursive subprogram control allows a subprogram to call itself.

This means that the subprogram can be executed repeatedly, with each call resulting in a new instance of the subprogram.

Recursive subprograms are useful for solving problems that can be broken down into smaller subproblems that can be solved using the same algorithm.

Recursive subprograms are also useful for solving problems that involve data structures such as trees, graphs, and linked lists.

A common example of a recursive subprogram is the calculation of the factorial of a number.

Recursive Subprogram

```
int factorial(int n) {
    if (n == 1) {
        return 1;
    } else {
        return n * factorial(n-1);
    }
}


int main(){
            /* Call subprogram
            result = factorial(5)
}
```

Questions?