C and Fortran 90

Computational Physics

Abhishek Bohare

a_bohare@icloud.com

typeset with LaTeX

Contents

1 Fortran 90/95 **1**

1.1 Fortran 90/95 1

1.1.1 Programming basics 1

1.1.2 Adding two numbers 1

1.1.3 Programmes and topics to cover 1

1.2 Logical operations 1

1.3 The `if` condition and the flowchart 2

1.4 Loops 3

1.4.1 The `do` loop 3

1.5 `if ... else` 4

1.6 Advantages and limitations of flowcharts 4

1.6.1 Advantages 4

1.6.2 Limitations 4

2 C Language **6**

2.1 The C language 6

2.1.1 Facts 6

2.1.2 Compiling from the terminal 6

2.1.3 Applications 7

2.1.4 Pre-requirements 7

2.1.5 A basic programme 7

2.2 Types in C 7

2.2.1 Integer types 7

2.2.2 Floating-point types 7

2.2.3 Variable types 8

2.3 Constants 8

2.4 Storage classes 8

2.5 Operators 8

2.6 Decision making 8

2.6.1 `if` 9
2.6.2 `if ... else` 9
2.7 Loops 9
2.7.1 `while` 9
2.7.2 `for` 10
2.7.3 `do while` 10
2.8 Arrays 10
2.9 Recursion 10
2.10 Input and output statements in C 11

Todo list

The first page of the notebook is headed “Unit 4, C programming”, but the text on it is about Fortran 90/95. 1

The last part of the second sentence about equipment control is hard to read (“for use in (equipment) alternatives available”); it is read here as the usual wording of the source guide. 1

The sticky note is partly hidden behind the page edge. Its last entry is not legible. 1

For `.not.` the example on the page reads “A and B is true”. It is read here as `.not.(A .and. B)`, which is true when `A` is true and `B` is false. 2

The handwritten version never gives `n` a value. The line `n = 10` was added so that the programme adds 1 to 10. 4

The ANSI C standard (ANSI X3.159) was ratified in 1989. ISO adopted it in 1990. 6

The Linux kernel is written mainly in C. MySQL is written in C and C++. 6

The function above has no exit condition, so it would call itself without end. A real recursive function needs a condition that stops the recursion. 11

The notes write that `stdout` “sends output to disk” and list `printf` and `scanf` under it. Standard output normally goes to the screen, `scanf` reads input (`stdin`), and `putchar` and `puts` are output functions. `getch()` and `getche()` are non-standard (`conio.h`). 11

1 Fortran 90/95

The first page of the notebook is headed "Unit 4, C programming", but the text on it is about Fortran 90/95.

1.1 Fortran 90/95

Fortran is one of the basic programming languages available. The name Fortran is short for *FORmula TRANslation*. This guide is based on Fortran 90, which is a version agreed in 1990. Fortran 95, a later standard, was a minor revision of Fortran 90. Fortran 2003 was in its final draft stage when the notes were written.

Fortran was developed for general scientific computing and is a very popular language for this purpose. In 1996 it was estimated that Fortran was employed for more than 90% of scientific computing. Fortran is not, however, particularly suitable as a non-scientific general-purpose language, or for use in equipment control, commerce or text management, for which alternatives are available.

Fortran 90 is available on the ITS Linux service, the UNIX service and on the networking PC service.

The last part of the second sentence about equipment control is hard to read ("for use in (equipment) alternatives available"); it is read here as the usual wording of the source guide.

1.1.1 Programming basics

A programme is simply a set of instructions that tell a computer to do a particular task. To create a programme that works, and works efficiently, you need to do the following before you start writing it:

- understand the aim;
- make sure you know how to compute the result.

1.1.2 Adding two numbers

A video of this programme was uploaded on YouTube.

1.1.3 Programmes and topics to cover

A sticky note in the margin lists the programmes and topics to do: matrix multiplication; the sum of values; the roots of a quadratic equation; input and output statements; `do while`; `if ... else`.

The sticky note is partly hidden behind the page edge. Its last entry is not legible.

1.2 Logical operations

Assume two logical variables, `A` = true and `B` = false. The logical operators of Fortran are written between dots.

Operator	Description	Example
`.and.`	Called the AND operator. If both operands are non-zero (true), the condition becomes true.	`A .and. B` is false
`.or.`	Called the OR operator. If any one operand is non-zero (true), the condition becomes true.	`A .or. B` is true
`.not.`	Called the NOT operator. It is used to reverse the value of its operand: if the operand is true, the NOT operator makes it false.	`.not.(A .and. B)` is true
`.eqv.`	Called the logical equivalent operator. It is used to check the equivalence of two logical values.	`A .eqv. B` is false
`.neqv.`	Called the logical non-equivalent operator. It is used to check the non-equivalence of two logical values.	`A .neqv. B` is true

For `.not.` the example on the page reads "A and B is true". It is read here as `.not.(A .and. B)`, which is true when `A` is true and `B` is false.

1.3 The `if` condition and the flowchart

Decision-making structures require that the programmer specifies one or more conditions to be evaluated or tested by the programme, along with a statement or statements to be executed if the condition is determined to be true and, optionally, other statements to be executed if the condition is determined to be false.

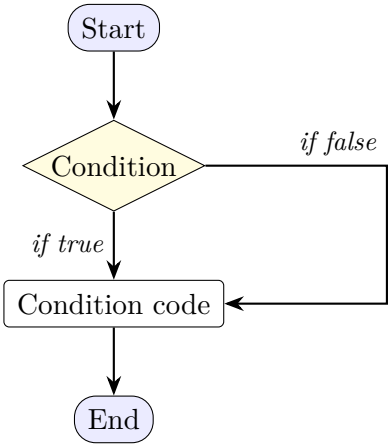


Figure 1.1: The flowchart of an `if` statement: the conditional code runs only when the condition is true.

Other forms: `if`, `if ... else`, nested `if`, `select`, nested `select`.

An `if` statement consists of a logical expression followed by one or more statements, and it is terminated by an `end if` statement.

Syntax:

```
if (logical_expression) then
    statement
end if
```

You can give a name to the `if` block. The syntax of a named `if` statement is

```
[name:] if (logical_statement) then
    ! various statements
    ...
end if [name]
```

1.4 Loops

There may be a situation when you need to execute a block of code several times. In general, statements are executed sequentially: the first statement in a function is executed first, then the second, and so on. Programming languages provide various control structures that allow for more complicated execution paths. A loop statement allows us to execute a statement or a group of statements multiple times.

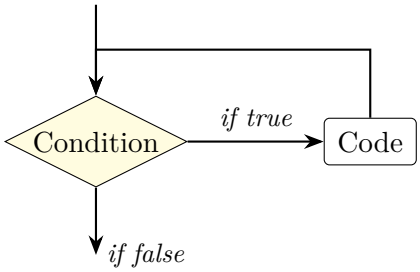


Figure 1.2: The flowchart of a loop: the code runs again and again while the condition is true.

Types: `do`, `do while` and nested loops.

1.4.1 The `do` loop

The `do` loop construct enables a statement, or a series of statements, to be carried out iteratively while a given condition is true.

```
do var = start, stop [, step]
    ! statement(s)
    ...
end do
```

Sum of the numbers 1 to 10 with a `do` loop

```
integer i, n, sum
n = 10
sum = 0
do 10 i = 1, n
    sum = sum + i
    write(*,*) 'i = ', i
    write(*,*) 'sum = ', sum
10 continue
```

The loop adds the values 1 to 10; changing `n` adds 1 to `n`, or 1 to 100, and so on.

The handwritten version never gives `n` a value. The line `n = 10` was added so that the programme adds 1 to 10.

1.5 `if ... else`

An `if ... then` statement can be followed by an optional `else` statement, which is executed when the logical expression is false.

Syntax:

```
if (logical_expression) then
    statement(s)
else
    other_statement(s)
end if
```

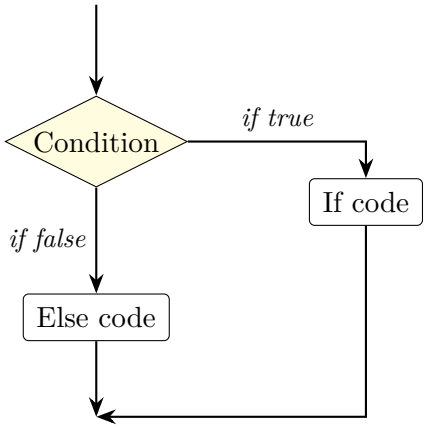


Figure 1.3: The flowchart of `if ... else`: one of the two blocks of code runs, and then the programme continues.

1.6 Advantages and limitations of flowcharts

1.6.1 Advantages

1. visual clarity;
2. instant communication;
3. effective coordination;
4. increase in efficiency;
5. effective analysis;
6. problem solving;
7. proper documentation.

1.6.2 Limitations

Flowcharts have many limitations:

1. complex logic;

2. alteration and modification;
3. reproduction;
4. the essentials of what is done can easily be lost in the technical details of how it is done;
5. factors that affect the sequence are not included.

2 C Language

2.1 The C language

C is a general-purpose, procedural, imperative computer programming language developed in 1972 by Dennis M. Ritchie at the Bell Telephone Laboratories to develop the UNIX operating system. C is one of the most widely used computer languages. It keeps fluctuating at the top of the popularity scale along with the Java programming language, which is equally popular and widely used among modern software programmers. The notes cite Codezclub and Tutorialspoint as sources.

Why is C so popular? Because it is

- easy to learn;
- a structured language;
- efficient: programmes are efficient;
- suited to low-level activity: a low-level language is one that provides little or no abstraction from a computer's instruction set architecture;
- compiled on a variety of computer platforms.

2.1.1 Facts

- C was invented to write the UNIX operating system.
- C is a successor of the B language, which was introduced in the 1970s.
- The language was formalised in 1988 by ANSI.
- The UNIX operating system was totally written in C.
- It is a very popular language. Today's most popular Linux operating system and the RDBMS MySQL have been written in C.

The ANSI C standard (ANSI X3.159) was ratified in 1989. ISO adopted it in 1990.

The Linux kernel is written mainly in C. MySQL is written in C and C++.

2.1.2 Compiling from the terminal

```
cd Desktop/
touch File.c
gcc File.c -o Out
./Out
```

The commands move to the Desktop folder, create the source file, compile it with the GNU compiler into an executable called `Out`, and run it.

2.1.3 Applications

C was initially used for system development work. Some examples of what is written in C are

1. operating systems;
2. language compilers;
3. assemblers;
4. text editors;
5. print spoolers;
6. network drivers;
7. modern programmes;
8. databases;
9. language interpreters;
10. utilities.

2.1.4 Pre-requirements

To write and run C you need two tools: a *text editor* and a *compiler*.

2.1.5 A basic programme

A C programme basically consists of the following parts:

Part	**Example**
preprocessor commands	`#include <stdio.h>`
functions	`int main()`
variables	
statements and expressions	(the mathematics of the programme)
comments	`/* ... */`

2.2 Types in C

Group	**Meaning**
Basic types	the integer types and the floating-point types
Enumerated types	arithmetic types used to define variables that can be assigned only certain discrete integer values
The type `void`	indicates that no value is available
Derived types	pointer types, array types, union types and function types

2.2.1 Integer types

`char`, `unsigned char`, `signed char`, `int`, `unsigned int`, `short`, `unsigned short`, `long`, `unsigned long`.

2.2.2 Floating-point types

`float`, `double`, `long double`.

2.2.3 Variable types

A variable is just a given name to a storage area. Its type is one of `char`, `int`, `float`, `double` or `void`.

2.3 Constants

Constants refer to fixed values that the programme may not alter during its execution. They can be defined with the preprocessor command `#define`.

2.4 Storage classes

A storage class defines the scope (visibility) and lifetime of variables and/or functions within a C programme.

`auto`	the default storage class
`register`	defines local variables that should be stored in a register instead of RAM
`static`	instructs the compiler to keep a local variable in existence during the lifetime of the programme
`extern`	a reference to a global variable that is visible to all the programme files

2.5 Operators

An operator is a symbol that tells a compiler to perform a specific mathematical or logical function.

Kind	**Operators**
Arithmetic	`+`, `-`, `*`, `/`, `%`, `++`, `--`
Relational	`==`, `!=`, `>`, `<`, `>=`, `<=`
Logical	`&&` (AND), `||` (OR), `!` (NOT)
Bitwise (work bit by bit)	`&`, `|`, `^` (XOR), `~` (complement), `«`, `»`
Assignment	`=`, `+=`, `-=`, `*=`, `/=`, `%=`, `«=`, `»=`, `&=`, `^=`, `|=`
Miscellaneous	`sizeof()`, `&`, `*`, `?:`

2.6 Decision making

A decision-making structure specifies one or more conditions to be evaluated or tested, and the statement or statements to be executed.

C assumes non-zero values as true and zero as false.

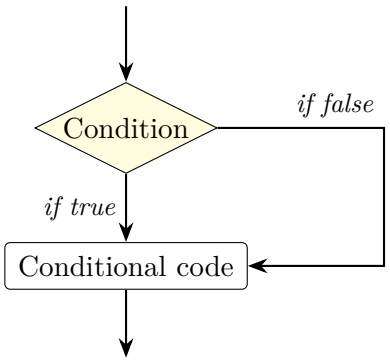


Figure 2.1: The flowchart of a decision.

2.6.1 `if`

An `if` statement consists of a Boolean expression followed by one or more statements.

```
if (boolean_expression)
{
    /* statements will execute if the boolean expression is true */
}
```

2.6.2 `if ... else`

```
if (boolean_expression)
{
    /* statements: the expression is true */
}
else
{
    /* statements: the expression is false */
}
```

2.7 Loops

A loop is used when a block of code needs to be executed several times: a loop statement allows us to execute a statement or a group of statements multiple times. The loop of Figure 2.2 repeats the code while the condition is true.

Types of loop: `while`, `for` and `do while`.

Loop control statements: `break`, `continue` and `goto`.

2.7.1 `while`

Repeatedly executes a target statement.

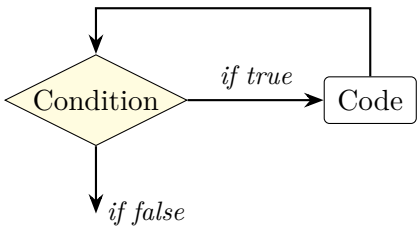


Figure 2.2: The flowchart of a loop.

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

2.7.2 `for`

A repetition control structure that allows us to write efficiently a loop that needs to execute a specific number of times.

2.7.3 `do while`

A `do while` loop is similar to a `while` loop, except that it is guaranteed to execute at least one time. It is used in scenarios where the loop should execute at least once.

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

2.8 Arrays

An array is a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

2.9 Recursion

Recursion is the process of repeating items in a self-similar way. In a programming language, if a programme allows you to call a function inside the same function, then it is called a recursion of a function.

```
void recursion()
{
```

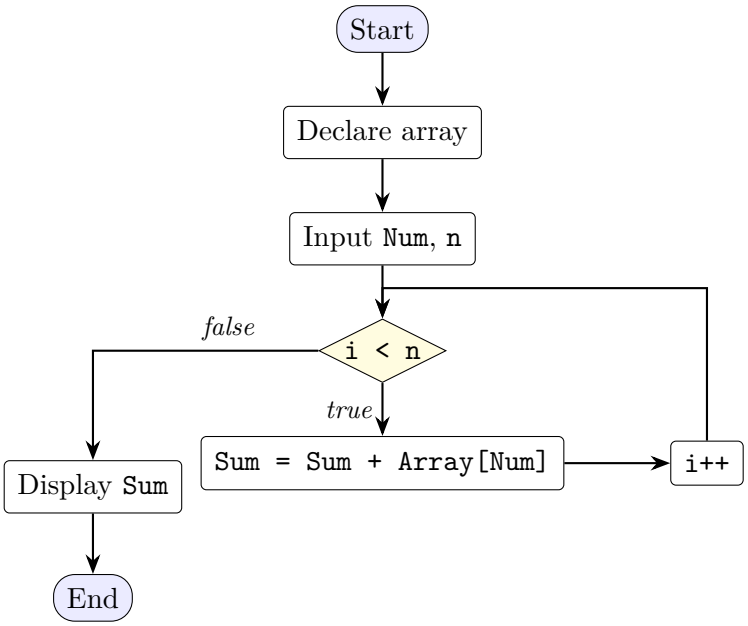


Figure 2.3: Adding up the elements of an array with a loop.

```
    recursion();      /* the function calls itself */
}

int main()
{
    recursion();
}
```

The C programming language supports recursion, that is, a function can call itself. But when using recursion, programmers need to be careful to define an exit condition from the function, otherwise it will go into an infinite loop.

Advantages:

- it reduces the unnecessary calling of functions;
- it is extremely useful when applying the same solution again and again.

The function above has no exit condition, so it would call itself without end. A real recursive function needs a condition that stops the recursion.

2.10 Input and output statements in C

C treats input and output as three standard files.

Standard file	Purpose	Functions
standard input (`stdin`)	receiving input	`getchar()`, `putchar()`, `gets()`, `puts()`, `getch()`, `getche()`
standard output (`stdout`)	sending output	`printf`, `scanf`
standard error (`stderr`)	displaying or storing errors	

The notes write that `stdout` "sends output to disk" and list `printf` and `scanf` under it. Standard output normally goes to the screen, `scanf` reads input (`stdin`), and `putchar` and `puts` are output functions. `getch()` and `getche()` are non-standard (`conio.h`).