

NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH DATA
ANALYTICS--CP09



SUBJECT CODE : CP00-01-04

SUBJECT TITLE : Problem Solving Techniques

Theory Hrs / week : 5

Total Hrs/semester : 100

Category : Information Technology

Revision No : 0

Sem:1

SL NO	MAJOR TOPICS	TIME ALLOATED
1	Introduction to Algorithm	10
1.1	What is an algorithm?	5
1.2	Characteristics of an Algorithm	5
2	Method for Developing an Algorithm	30
2.1	sequence structure	5
2.2	selection structure	5
2.3	repetition structure	5
2.4	Flowcharts	5
2.5	Flowchart Constructs	5
2.6	Flowchart Example	5
3	Pseudocode (or Program Design Language)	12
3.1	Pseudocode Language Constructs	2
3.2	Computation/Assignment	2
3.3	Input/Output	2
3.4	Selection	2

NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH
DATA ANALYTICS--CP09

3.5	Repetition	2
3.6	Pseudocode Example	2
4	Programming using C	20
4.1	How to develop a program using C language	10
4.2	Simple program examples	10
5	Problem solving	30
5.1	Introducing Computational Thinking	3
5.2	What is CT?	3
5.3	Exploring Algorithms	3
5.4	Computational Thinking is a problem-solving process that includes the following characteristics.	3
5.5	Finding Patterns	3
5.6	Developing Algorithms	3
5.7	Decomposition	2
5.8	Abstraction	2
5.9	Pattern Recognition	2
5.10	Designing algorithms	2
5.11	Computer Hardware vs Computer Software	2
5.12	Final Project: Applying Computational Thinking	2

CHAPTER 1

Introduction to Algorithm

1.1 What is an algorithm?

An **algorithm** (pronounced AL-go-rith-um) is a procedure or formula for solving a problem, based on conducting a sequence of specified actions. A computer program can be viewed as an elaborate **algorithm**. In mathematics and computer science, an **algorithm** usually means a small procedure that solves a recurrent problem.

1.2 Characteristics of an Algorithm

- ☐ Well-ordered: the steps are in a clear order
- ☐ Unambiguous: the operations described are understood by a computing agent without further simplification
- ☐ Effectively computable: the computing agent can actually carry out the operation

2 Method for Developing an Algorithm

1. Define the problem: State the problem you are trying to solve in *clear* and *concise* terms.
2. List the *inputs* (information needed to solve the problem) and the *outputs* (what the algorithm will produce as a result)
3. Describe the steps needed to convert or manipulate the inputs to produce the outputs. Start at a high level first, and keep refining the steps until they are *effectively computable* operations.
4. Test the algorithm: choose data sets and verify that your algorithm works

Structured Programming

☐ In 1966, computer scientists Corrado Böhm and Giuseppe Jacopini demonstrated that all programs could be written using three control structures: Sequence, Selection, and Repetition

2.1 sequence structure

☐ The *sequence* structure is the construct where one statement is executed after another

2.2 selection structure

☐ The *selection* structure is the construct where statements can be executed or skipped depending on whether a condition evaluates to TRUE or FALSE

☐ There are three selection structures in C:

1. IF
2. IF – ELSE
3. SWITCH





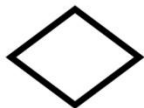

2.3 repetition structure

- The *repetition* structure is the construct where statements can be executed repeatedly until a condition evaluates to TRUE or FALSE
- There are three repetition structures in C:
 1. WHILE
 2. DO – WHILE
 3. FOR

2.4 Flowcharts

Flowcharts

- A graphical tool that *diagrammatically* depicts the steps and structure of an algorithm or program
- Symbols (the most commonly used ones)

Symbol	Name/Meaning	Symbol	Meaning
	<u>Process</u> – Any type of internal operation: data transformation, data movement, logic operation, etc.		<u>Connector</u> – connects sections of the flowchart, so that the diagram can maintain a smooth, linear flow
	<u>Input/Output</u> – input or output of data		<u>Terminal</u> – indicates start or end of the program or algorithm
	<u>Decision</u> – evaluates a condition or statement and branches depending on whether the evaluation is true or false		<u>Flow lines</u> – <i>arrows</i> that indicate the direction of the progression of the program

NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH
DATA ANALYTICS--CP09

- ☐ General rules for flowcharts
- ☐ All symbols of the flowchart are connected by flow lines (note *arrows*, not lines)
- ☐ Flowlines enter the top of the symbol and exit out the bottom, except for the Decision symbol, which can have flow lines exiting from the bottom or the sides
- ☐ Flowcharts are drawn so flow generally goes from top to bottom
- ☐ The beginning and the end of the flowchart is indicated using the Terminal symbol

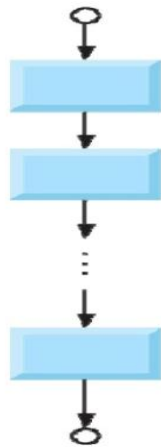
2.5 Flowchart Constructs

The flowchart equivalents for the structured programming constructs described earlier are:

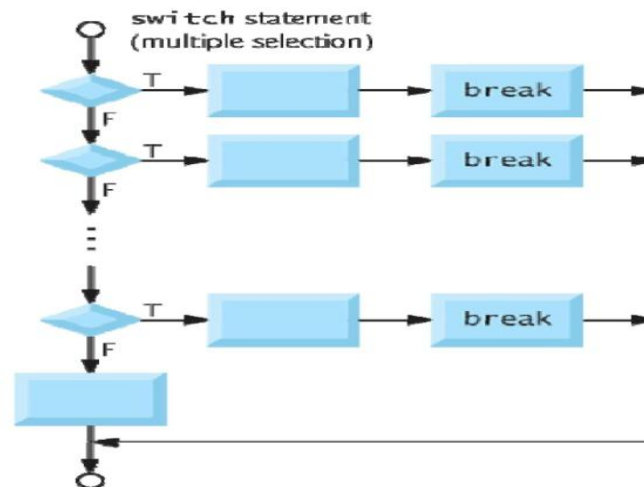
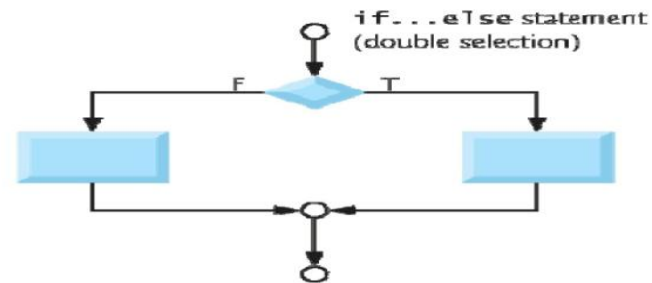
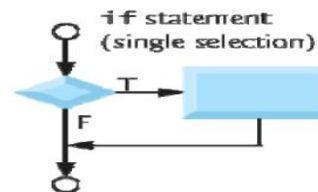
NETTUR TECHNICAL TRAINING FOUNDATION

COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH DATA ANALYTICS--CP09

Sequence

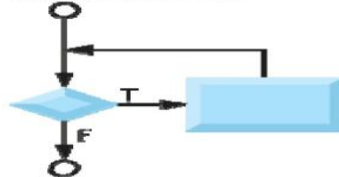


Selection

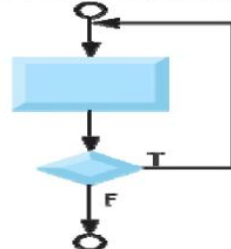


Repetition

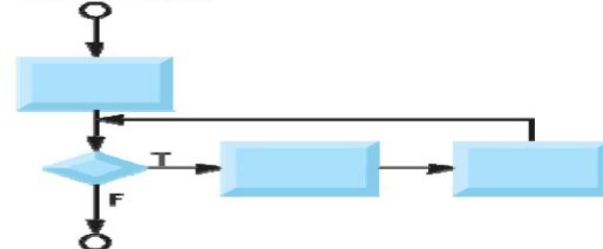
while statement



do...while statement



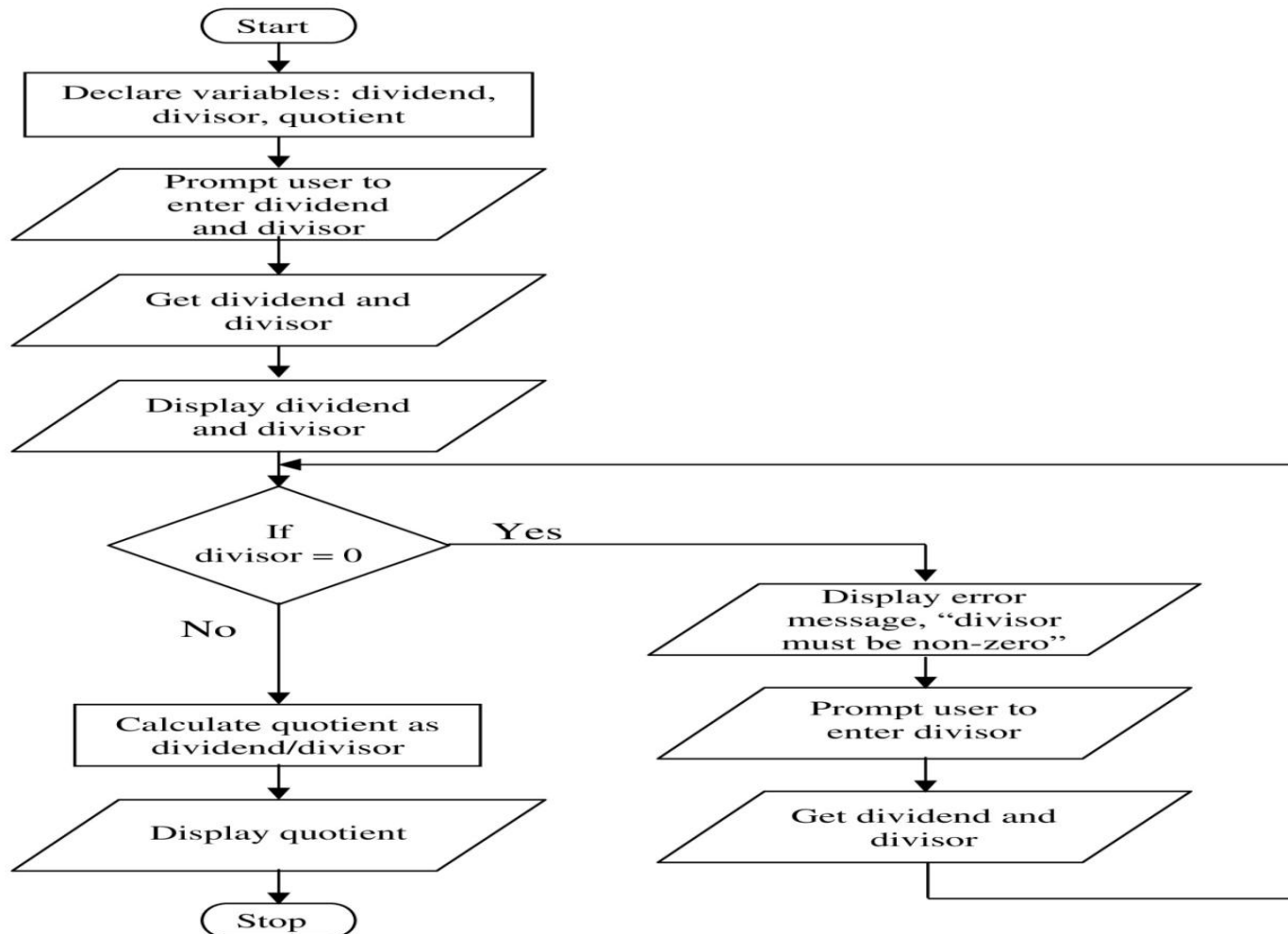
for statement



NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH
DATA ANALYTICS--CP09

Flowchart Example

Express an algorithm to get two numbers from the user (dividend and divisor), testing to make sure that the divisor number is not zero, and displaying their quotient using a *flowchart*.



3 Pseudocode (or Program Design Language)

- Consists of natural language-like statements that precisely describe the steps of an algorithm or program
- Statements describe *actions*

NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH
DATA ANALYTICS--CP09

- ☐ Focuses on the *logic* of the algorithm or program
- ☐ Avoids language-specific elements
- ☐ Written at a level so that the desired programming code can be generated almost automatically from each statement
- ☐ Steps are numbered. Subordinate numbers and/or indentation are used for dependent statements in selection and repetition structures.

3.1 Pseudocode Language Constructs

3.2 Computation/Assignment

- ☐ **Compute** var1 as the sum of x and y
- ☐ **Assign** expression to var2
- ☐ **Increment** counter1

3.3 Input/Output

- ☐ Input: **Get** var1, var2, ...
- ☐ Output: **Display** var1, var2, ...

3.4 Selection

Single-Selection IF

1. **IF** condition **THEN** (IF condition is true, then do subordinate statement 1, etc. If condition is false, then skip statements)

1.1 statement 1

1.2 etc.

Double-Selection IF

2. **IF** condition **THEN** (IF condition is true, then do subordinate statement 1, etc. If

condition is false, then skip statements and execute statements under ELSE)

2.1 statement 1

2.2 etc.

3. **ELSE** (else if condition is not true, then do subordinate statement 2, etc.)

3.1 statement 2

3.2 statement 3

4. **SWITCH** expression TO

4.1 case 1: action1

4.2 case 2: action2

4.3 etc.

4.4 default: actionx

3.5 Repetition

5. **WHILE** condition (while condition is true, then do subordinate statements)

5.1 statement 1

5.2 etc.

DO – WHILE structure (like WHILE, but tests condition at the *end* of the loop. Thus, statements in the structure will always be executed at least once.)

6. **DO**

6.1 statement 1

6.2 etc.

7. **WHILE** condition

FOR structure (a specialized version of WHILE for repeating execution of statements a specific number of times)

8. **FOR** bounds on repetition

- 8.1 statement 1
- 8.2 etc.

3.6 Pseudocode Example

Express an algorithm to get two numbers from the user (dividend and divisor), testing to make sure that the divisor number is not zero, and displaying their quotient using *pseudocode*

1. Declare variables: dividend, divisor, quotient
2. Prompt user to enter dividend and divisor
3. Get dividend and divisor
4. IF divisor is equal to zero, THEN
 - 4.1. DO
 - 4.1.1. Display error message, “divisor must be non-zero”
 - 4.1.2. Prompt user to enter divisor
 - 4.1.3. Get divisor
 - 4.2. WHILE divisor is equal to zero
5. ENDIF
6. Display dividend and divisor
7. Calculate quotient as dividend/divisor
8. Display quotient

3 Programming using C

History

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.

The C Language is developed for creating system applications that directly interact with the hardware devices such as drivers, kernels, etc.

C programming is considered as the base for other programming languages, that is why it is known as mother language.

It can be defined by the following ways:

- 1 Mother language
- 2 System programming language
- 3 Procedure-oriented programming language
- 4 Structured programming language
- 5 Mid-level programming language

1) C as a mother language

C language is considered as the mother language of all the modern programming languages because **most of the compilers, JVMs, Kernels, etc. are written in C language**, and most of the programming languages follow C syntax, for example, C++, Java, C#, etc.

It provides the core concepts like the array, strings, functions, file handling, etc. that are being used in many languages like C++, Java, C#, etc.

2) C as a system programming language

A system programming language is used to create system software. C language is a system programming language because it **can be used to do low-level programming (for example driver and kernel)**. It is generally used to create hardware devices, OS, drivers, kernels, etc. For example, Linux kernel is written in C.

It can't be used for internet programming like Java, .Net, PHP, etc.

3) C as a procedural language

A procedure is known as a function, method, routine, subroutine, etc. A procedural language **specifies a series of steps for the program to solve the problem.**

A procedural language breaks the program into functions, data structures, etc.

C is a procedural language. In C, variables and function prototypes must be declared before being used.

4) C as a structured programming language

A structured programming language is a subset of the procedural language. **Structure means to break a program into parts or blocks** so that it may be easy to understand.

In the C language, we break the program into parts using functions. It makes the program easier to understand and modify.

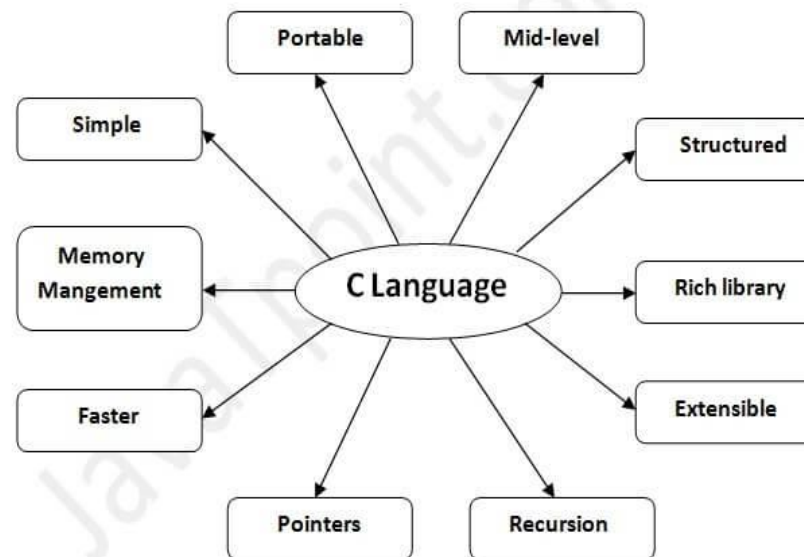
5) C as a mid-level programming language

C is considered as a middle-level language because it **supports the feature of both low-level and high-level languages**. C language program is converted into assembly code, it supports pointer arithmetic (low-level), but it is machine independent (a feature of high-level).

A **Low-level language** is specific to one machine, i.e., machine dependent. It is machine dependent, fast to run. But it is not easy to understand.

A **High-Level language** is not specific to one machine, i.e., machine independent. It is easy to understand.

Features of C Language



JavaTpoint.com

C is the widely used language. It provides many **features** that are given below.

1. Simple

2. Machine Independent or Portable
3. Mid-level programming language
4. structured programming language
5. Rich Library
6. Memory Management
7. Fast Speed
8. Pointers
9. Recursion
10. Extensible

How to install C

There are many compilers available for c and c++. You need to download any one. Here, we are going to use **Turbo C++**. It will work for both C and C++. To install the Turbo C software, you need to follow following steps.

1. Download Turbo C++
2. Create turboc directory inside c drive and extract the tc3.zip inside c:\turboc
3. Double click on install.exe file
4. Click on the tc application file located inside c:\TC\BIN to write the c program

1) Download Turbo C++ software

You can download turbo c++ from many sites. [download Turbo c++](#)

2) Create turboc directory in c drive and extract the tc3.zip

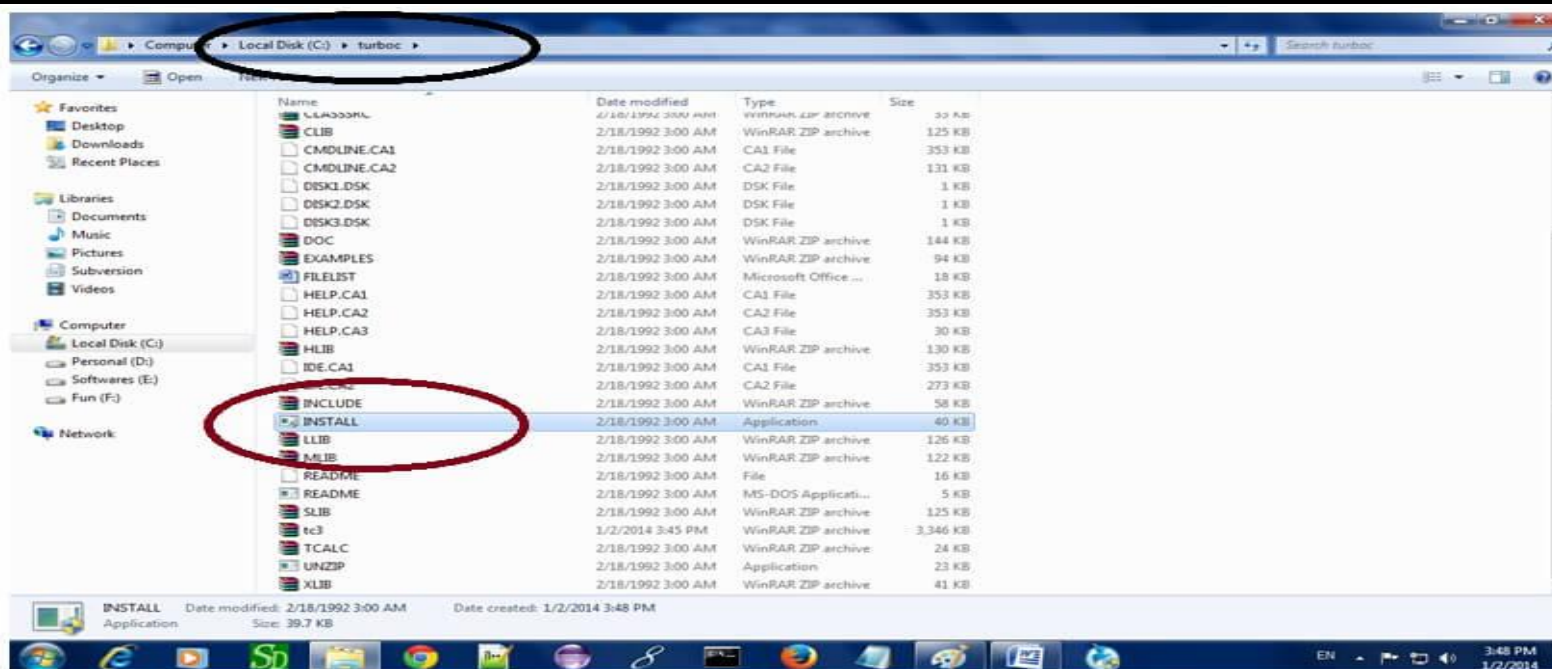
Now, you need to create a new directory turboc inside the c: drive. Now extract the tc3.zip file in c:\truboc directory.

3) Double click on the install.exe file and follow steps

Now, click on the install icon located inside the c:\turboc

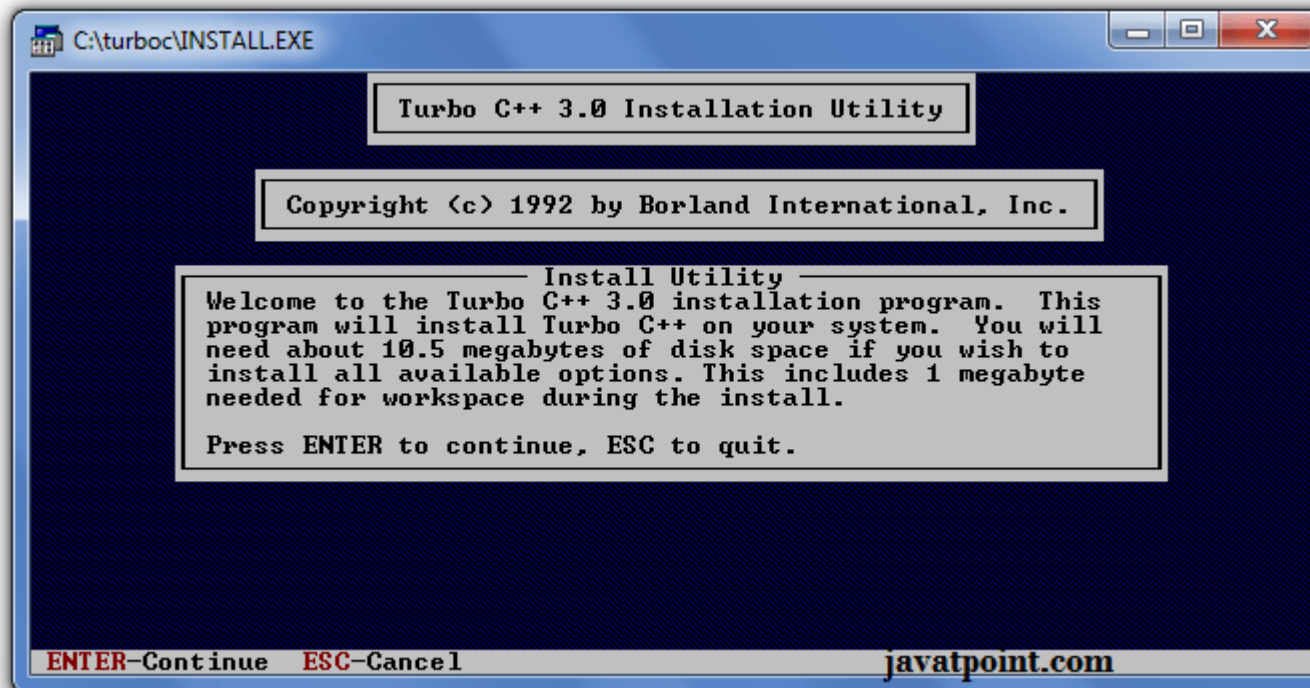
NETTUR TECHNICAL TRAINING FOUNDATION

COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH DATA ANALYTICS--CP09

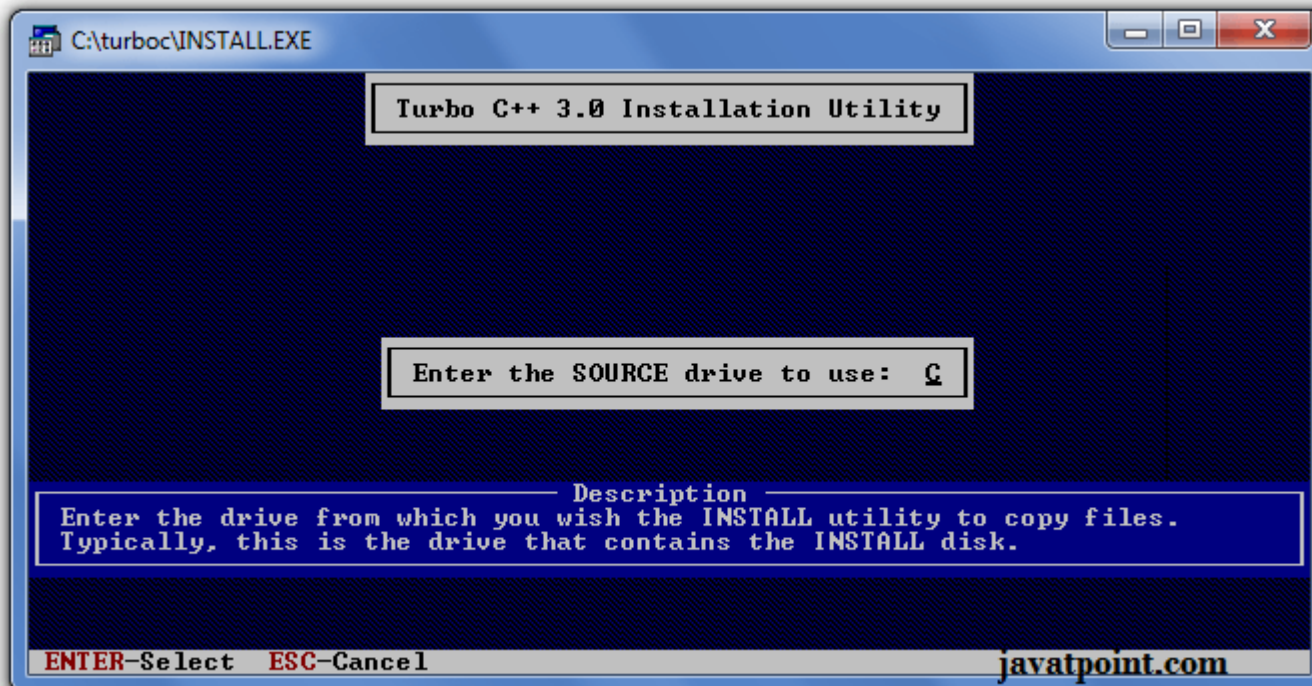


NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH
DATA ANALYTICS--CP09

It will ask you to install c or not, press enter to install.

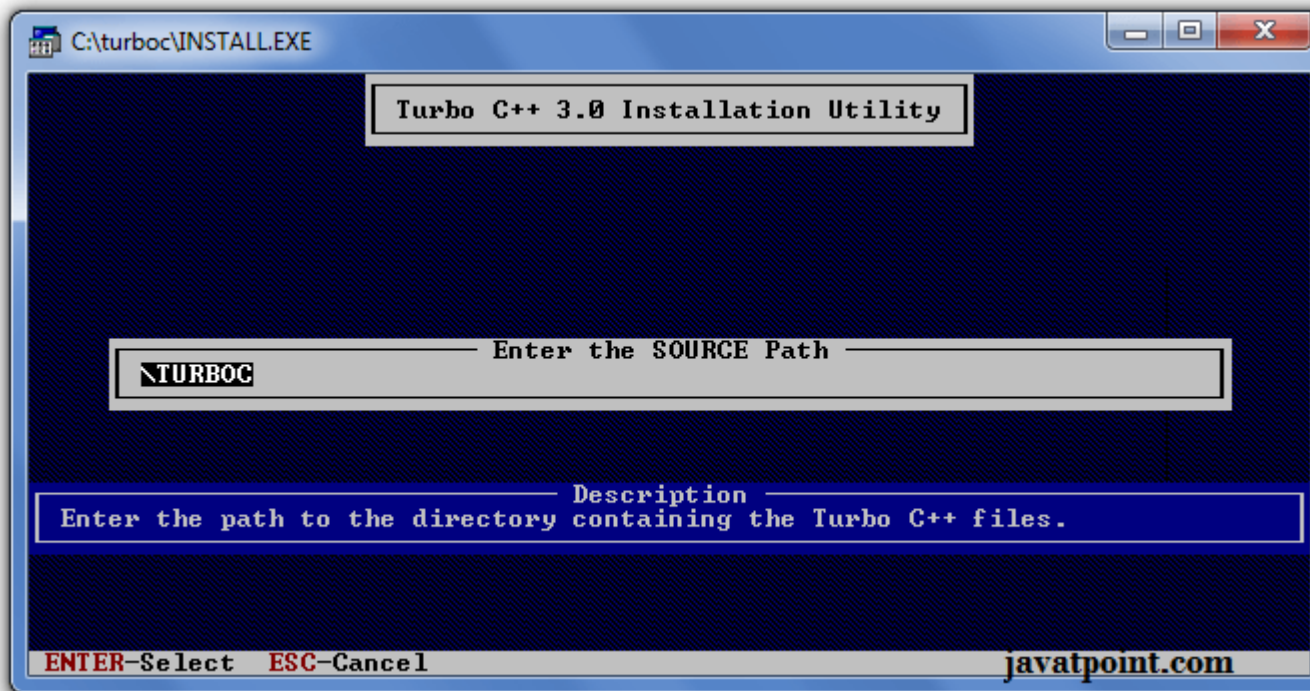


Change your drive to c, press c.



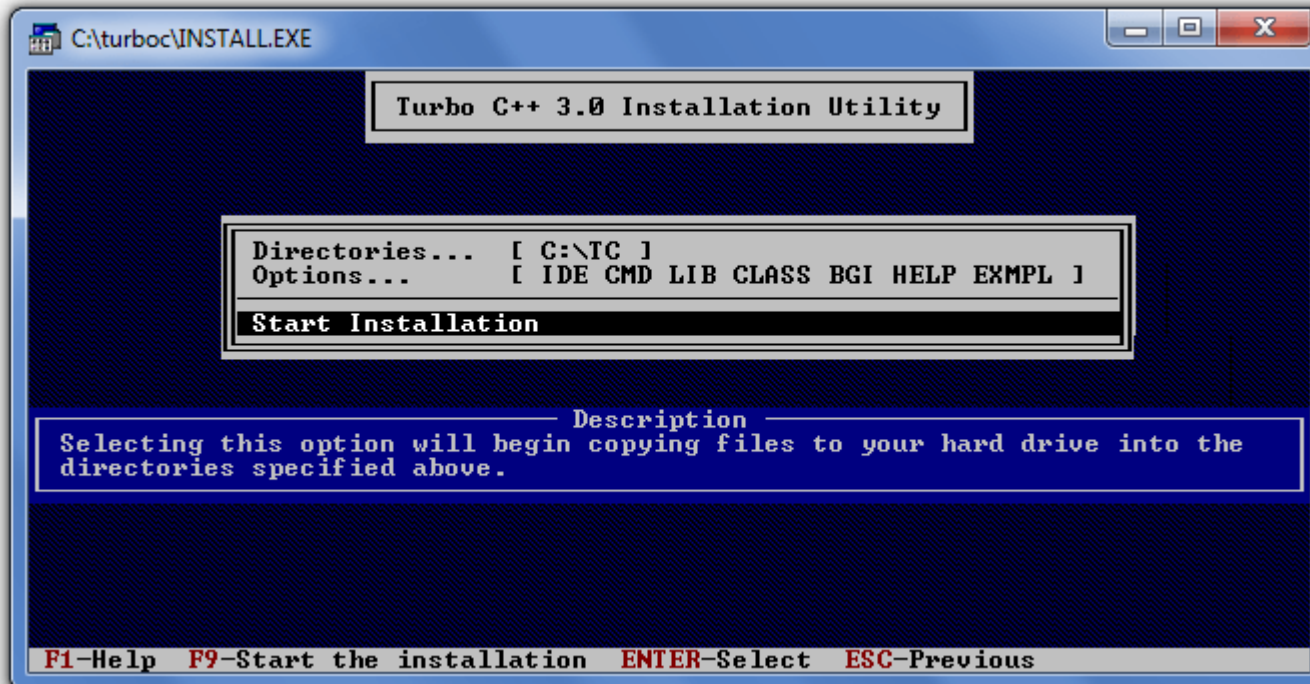
NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH
DATA ANALYTICS--CP09

Press enter, it will look inside the c:\turbo directory for the required files.



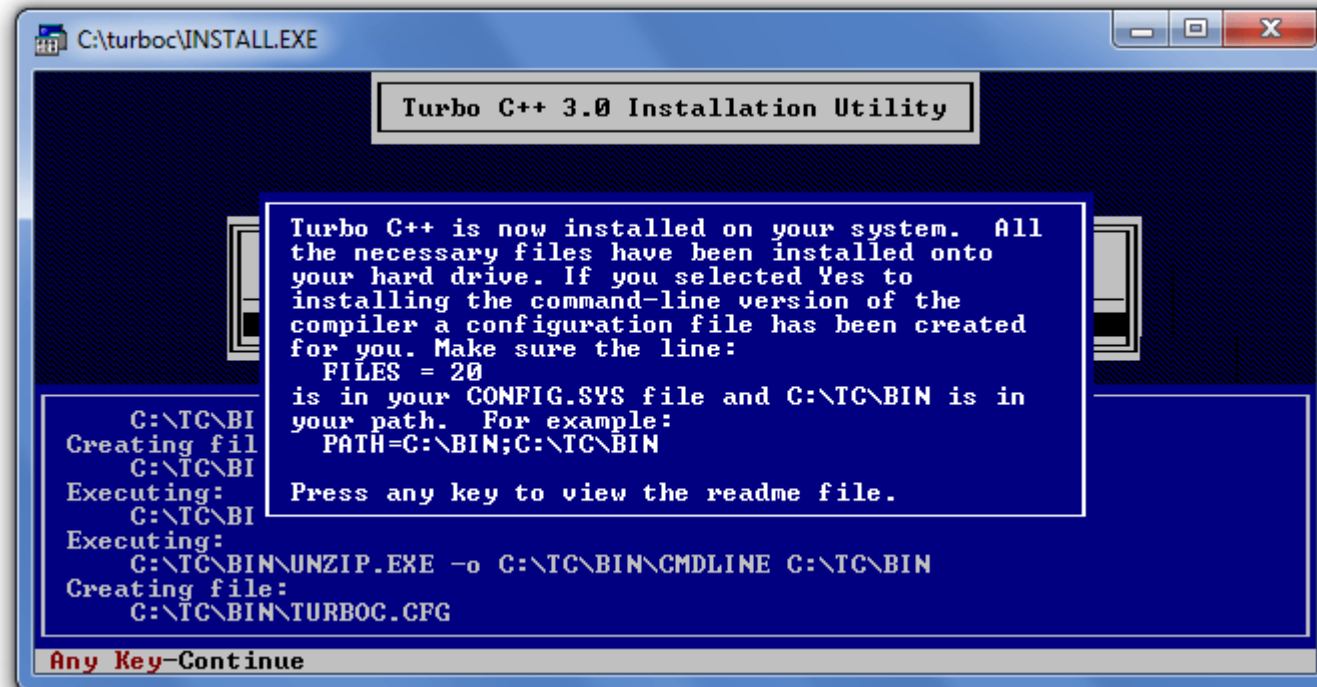
NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH DATA
ANALYTICS--CP09

Select Start installation by the down arrow key then press enter.



NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH
DATA ANALYTICS--CP09

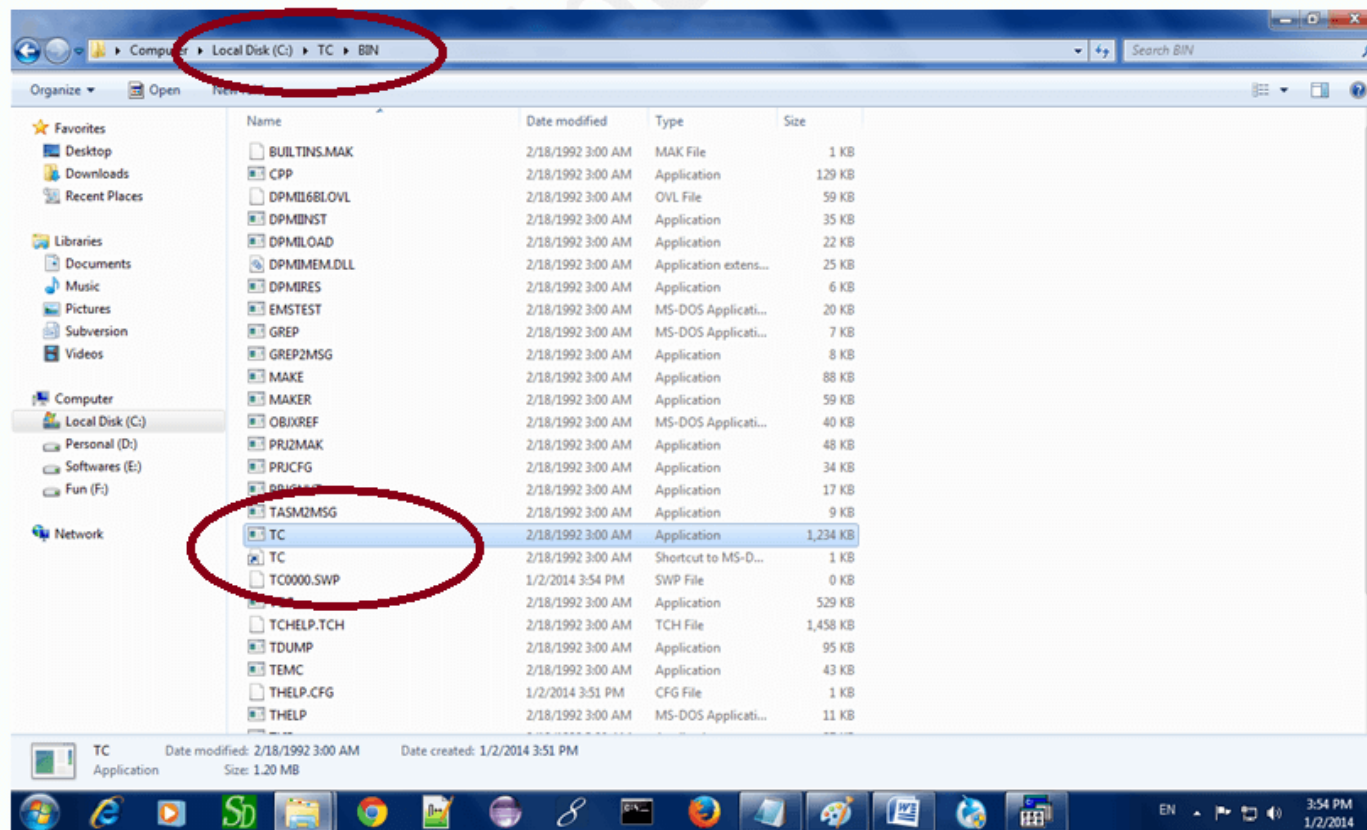
Now C is installed, press enter to read documentation or close the



software.

4) Click on the tc application located inside c:\TC\BIN

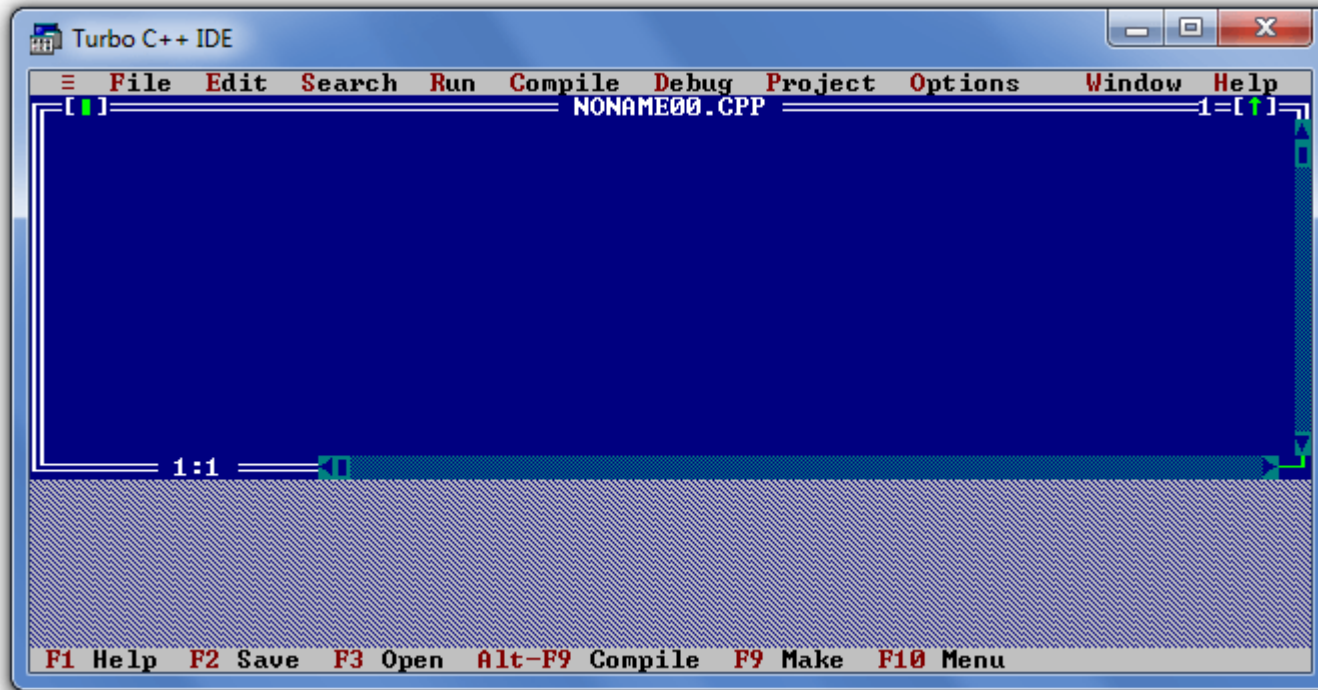
Now double click on the tc icon located in c:\TC\BIN directory to write the c program.



NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH
DATA ANALYTICS--CP09

In windows 7 or window 8, it will show a dialog block to ignore and close the application because fullscreen mode is not supported. Click on Ignore button.

Now it will showing following console.



C Program

In this tutorial, all C programs are given with C compiler so that you can quickly change the C program code.

NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH
DATA ANALYTICS--CP09

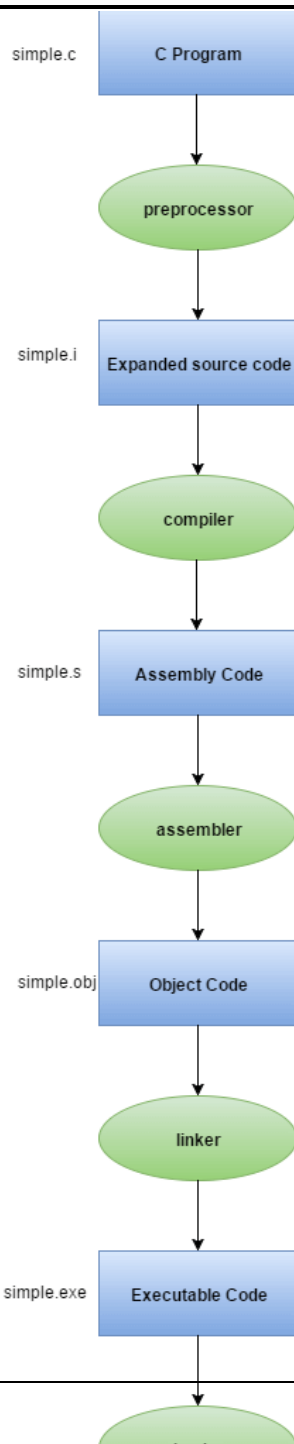
File: main.c

```
1. #include <stdio.h>
2. int main() {
3. printf("Hello C Programming\n");
4. return 0;
5. }
```

Execution Flow

NETTUR TECHNICAL TRAINING FOUNDATION

COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH DATA ANALYTICS--CP09



9 VERSION

DATED:1st JUNE 2019

Let's try to understand the flow of above program by the figure given below.

- 1) C program (source code) is sent to preprocessor first. The preprocessor is responsible to convert preprocessor directives into their respective values. The preprocessor generates an expanded source code.
- 2) Expanded source code is sent to compiler which compiles the code and converts it into assembly code.
- 3) The assembly code is sent to assembler which assembles the code and converts it into object code. Now a simple.obj file is generated.
- 4) The object code is sent to linker which links it to the library such as header files. Then it is converted into executable code. A simple.exe file is generated.
- 5) The executable code is sent to loader which loads it into memory and then it is executed. After execution, output is sent to console.

Variable Definition in C

A variable definition tells the compiler where and how much storage to create for the variable. A variable definition specifies a data type and contains a list of one or more variables of that type as follows –

```
type variable_list;
```

Here, **type** must be a valid C data type including char, w_char, int, float, double, bool, or any user-defined object; and **variable_list** may consist of one or more identifier names separated by commas. Some valid declarations are shown here –

```
int   i, j, k;  
char  c, ch;  
float f, salary;  
double d;
```

The line **int i, j, k;** declares and defines the variables i, j, and k; which instruct the compiler to create variables named i, j and k of type int.

Variables can be initialized (assigned an initial value) in their declaration. The initializer consists of an equal sign followed by a constant expression as follows –

```
type variable_name = value;
```

Some examples are –

```
extern int d = 3, f = 5;  // declaration of d and f.  
int d = 3, f = 5;        // definition and initializing d and f.  
byte z = 22;             // definition and initializes z.  
char x = 'x';            // the variable x has the value 'x'.
```

For definition without an initializer: variables with static storage duration are implicitly initialized with NULL (all bytes have the value 0); the initial value of all other variables are undefined.

Variable Declaration in C

A variable declaration provides assurance to the compiler that there exists a variable with the given type and name so that the compiler can proceed for further compilation without requiring the complete detail about the variable. A variable definition has its meaning at the time of compilation only, the compiler needs actual variable definition at the time of linking the program.

A variable declaration is useful when you are using multiple files and you define your variable in one of the files which will be available at the time of linking of the program. You will use the keyword **extern** to declare a variable at any place. Though you can declare a variable multiple times in your C program, it can be defined only once in a file, a function, or a block of code.

Example

Try the following example, where variables have been declared at the top, but they have been defined and initialized inside the main function –


```
#include <stdio.h>
```

```
// Variable declaration:
```

```
extern int a, b;
```

```
extern int c;
```

```
extern float f;
```

```
int main () {
```

```
    /* variable definition: */
```

```
    int a, b;
```

```
    int c;
```

```
    float f;
```

NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH
DATA ANALYTICS--CP09

```
/* actual initialization */  
  
a = 10;  
  
b = 20;  
  
  
c = a + b;  
printf("value of c : %d \n", c);  
  
  
f = 70.0/3.0;  
printf("value of f : %f \n", f);  
  
  
return 0;  
}
```

When the above code is compiled and executed, it produces the following result –

value of c : 30
value of f : 23.333334

The same concept applies on function declaration where you provide a function name at the time of its declaration and its actual definition can be given anywhere else. For example –

```
// function declaration
```

```
int func();
```

```
int main() {
```

```
    // function call
```

```
    int i = func();
```

```
}
```

```
// function definition
```

```
int func() {  
    return 0;  
}
```

C Input Output (I/O)

In this tutorial, you will learn to use scanf() function to take input from the user, and printf() function to display output to the user.

C Output

In C programming, printf() is one of the main output function. The function sends formatted output to the screen. For example,

Example 1: C Output

```
1. #include <stdio.h>  
2. int main()  
3. {  
4.     // Displays the string inside quotations
```

```
5. printf("C Programming");  
6. return 0;  
7. }
```

Output

C Programming

How does this program work?

- All valid C programs must contain the `main()` function. The code execution begins from the start of the `main()` function.
- The `printf()` is a library function to send formatted output to the screen. The function prints the string inside quotations.
- To use `printf()` in our program, we need to include `stdio.h` header file using `#include <stdio.h>` statement.
- The `return 0;` statement inside the `main()` function is the "Exit status" of the program. It's optional.

Example 2: Integer Output

```
1. #include <stdio.h>
2. int main()
3. {
4.     int testInteger = 5;
5.     printf("Number = %d", testInteger);
6.     return 0;
7. }
```

Output

Number = 5

We use %d format specifier to print int types. Here, the %d inside the quotations will be replaced by the value of testInteger.

Example 3: float and double Output

```
1. #include <stdio.h>
2. int main()
```

```
3. {  
4.   float number1 = 13.5;  
5.   double number2 = 12.4;  
6.  
7.   printf("number1 = %f\n", number1);  
8.   printf("number2 = %lf", number2);  
9.   return 0;  
10. }
```

Output

```
number1 = 13.500000  
number2 = 12.400000
```

To print float, we use %f format specifier. Similarly, we use %lf to print double values.

Example 4: Print Characters

```
1. #include <stdio.h>  
2. int main()  
3. {  
4.   char chr = 'a';
```

```
5. printf("character = %c.", chr);  
6. return 0;  
7. }
```

Output

character = a

To print char, we use %c format specifier.

C Input

In C programming, `scanf()` is one of the commonly used function to take input from the user. The `scanf()` function reads formatted input from the standard input such as keyboards.

Example 5: Integer Input/Output

```
1. #include <stdio.h>  
2. int main()  
3. {
```



```
4.  int testInteger;  
5.  printf("Enter an integer: ");  
6.  scanf("%d", &testInteger);  
7.  printf("Number = %d",testInteger);  
8.  return 0;  
9. }
```

Output

```
Enter an integer: 4  
Number = 4
```

Here, we have used %d format specifier inside the scanf() function to take int input from the user. When the user enters an integer, it is stored in the testInteger variable.

Notice, that we have used &testInteger inside scanf(). It is because &testInteger gets the address of testInteger, and the value entered by the user is stored in that address.

Example 6: Float and Double Input/Output

```
1. #include <stdio.h>  
2. int main()  
3. {
```

NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH
DATA ANALYTICS--CP09

```
4. float num1;
5. double num2;
6.
7. printf("Enter a number: ");
8. scanf("%f", &num1);
9. printf("Enter another number: ");
10. scanf("%lf", &num2);
11.
12. printf("num1 = %f\n", num1);
13. printf("num2 = %lf", num2);
14.
15. return 0;
16. }
```

Output

```
Enter a number: 12.523
Enter another number: 10.2
num1 = 12.523000
num2 = 10.200000
```

We use %f and %lf format specifier for float and double respectively.

Example 7: C Character I/O

```
1. #include <stdio.h>
2. int main()
3. {
4.     char chr;
5.     printf("Enter a character: ");
6.     scanf("%c",&chr);
7.     printf("You entered %c.", chr);
8.     return 0;
9. }
```

Output

```
Enter a character: g
You entered g.
```

When a character is entered by the user in the above program, the character itself is not stored. Instead, an integer value (ASCII value) is stored.

And when we display that value using `%c` text format, the entered character is displayed. If we use `%d` to display the character, it's ASCII value is printed.

Example 8: ASCII Value

```
1. #include <stdio.h>
2. int main()
3. {
4.     char chr;
5.     printf("Enter a character: ");
6.     scanf("%c", &chr);
7.
8.     // When %c is used, a character is displayed
9.     printf("You entered %c.\n", chr);
10.
11.     // When %d is used, ASCII value is displayed
12.     printf("ASCII value is % d.", chr);
13.     return 0;
14. }
```

Output

Enter a character: g
You entered g.
ASCII value is 103.

I/O Multiple Values

Here's how you can take multiple inputs from the user and display them.

```
1. #include <stdio.h>
2. int main()
3. {
4.     int a;
5.     float b;
6.
7.     printf("Enter integer and then a float: ");
8.
9.     // Taking multiple inputs
10.    scanf("%d%f", &a, &b);
11.
12.    printf("You entered %d and %f", a, b);
```

```
13.     return 0;  
14.     }
```

Output

Enter integer and then a float: -3
3.4
You entered -3 and 3.400000

Format Specifiers for I/O

As you can see from the above examples, we use

- 1 %d for int
- 2 %f for float
- 3 %lf for double
- 4 %c for char

Here's a list of commonly used C data types and their format specifiers.

NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH DATA
ANALYTICS--CP09



Data Type	Format Specifier
int	%d
char	%c
float	%f
double	%lf
short int	%hd
unsigned int	%u
long int	%li
long long int	%lli
unsigned long int	%lu
unsigned long long int	%llu
signed char	%c

Data Type	Format Specifier
unsigned char	%c
long double	%Lf

C - Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators –

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

We will, in this chapter, look into the way each operator works.

Arithmetic Operators

The following table shows all the arithmetic operators supported by the C language. Assume variable **A** holds 10 and variable **B** holds 20 then –

Operator	Description	Example
+	Adds two operands.	$A + B = 30$
–	Subtracts second operand from the first.	$A - B = -10$
*	Multiplies both operands.	$A * B = 200$
/	Divides numerator by de-numerator.	$B / A = 2$
%	Modulus Operator and remainder of after an integer division.	$B \% A = 0$
++	Increment operator increases the integer value by one.	$A++ = 11$

NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH
DATA ANALYTICS--CP09

--	Decrement operator decreases the integer value by one.	A-- = 9
----	--------------------------------------------------------	---------

Relational Operators

The following table shows all the relational operators supported by C. Assume variable **A** holds 10 and variable **B** holds 20 then –

Show Examples

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition	(A > B) is not true.

	becomes true.	
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(A <= B) is true.

Logical Operators

Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0, then –

Show Examples

Operator	Description	Example
----------	-------------	---------

NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH
DATA ANALYTICS--CP09

&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A B) is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ is as follows –

p	q	p & q	p q	p ^ q
0	0	0	0	0

NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH DATA
ANALYTICS--CP09



0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Assume $A = 60$ and $B = 13$ in binary format, they will be as follows –

$A = 0011\ 1100$

$B = 0000\ 1101$

$A \& B = 0000\ 1100$

$A | B = 0011\ 1101$

$A \wedge B = 0011\ 0001$

$\sim A = 1100\ 0011$

The following table lists the bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13, then –

NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH
DATA ANALYTICS--CP09

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
	Binary OR Operator copies a bit if it exists in either operand.	(A B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = ~(60), i.e., - 0111101
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000

>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111
----	---------------------------------------------------------------------------------------------------------------------------	--------------------------------

Assignment Operators

The following table lists the assignment operators supported by the C language –

Show Examples

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and	C -= A is equivalent to C = C - A

NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH
DATA ANALYTICS--CP09

	assigns the result to the left operand.	
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	$C *= A$ is equivalent to $C = C * A$
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	$C /= A$ is equivalent to $C = C / A$
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	$C \% = A$ is equivalent to $C = C \% A$
<<=	Left shift AND assignment operator.	$C <<= 2$ is same as $C = C << 2$
>>=	Right shift AND assignment operator.	$C >>= 2$ is same as C

NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH DATA
ANALYTICS--CP09



		$= C \gg 2$
$\&=$	Bitwise AND assignment operator.	$C \&= 2$ is same as $C = C \& 2$
$\wedge=$	Bitwise exclusive OR and assignment operator.	$C \wedge= 2$ is same as $C = C \wedge 2$
$ =$	Bitwise inclusive OR and assignment operator.	$C = 2$ is same as $C = C 2$

Misc Operators

↪ sizeof & ternary

Besides the operators discussed above, there are a few other important operators including **sizeof** and **? :** supported by the C Language.

Show Examples

Operator	Description	Example
----------	-------------	---------

NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH
DATA ANALYTICS--CP09

sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
? :	Conditional Expression.	If Condition is true ? then value X : otherwise value Y

Operators Precedence in C

Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

For example, $x = 7 + 3 * 2$; here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with $3*2$ and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Show Examples

Category	Operator	Associativity
Postfix	() [] -> . ++ --	Left to right
Unary	+ - ! ~ ++ -- (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right

NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH
DATA ANALYTICS--CP09

Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR		Left to right
Logical AND	&&	Left to right
Logical OR		Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= =	Right to left

Comma	,	Left to right
-------	---	---------------

Conditional statements

C if else Statement

The if-else statement in C is used to perform the operations based on some specific condition. The operations specified in if block are executed if and only if the given condition is true.

There are the following variants of if statement in C language.

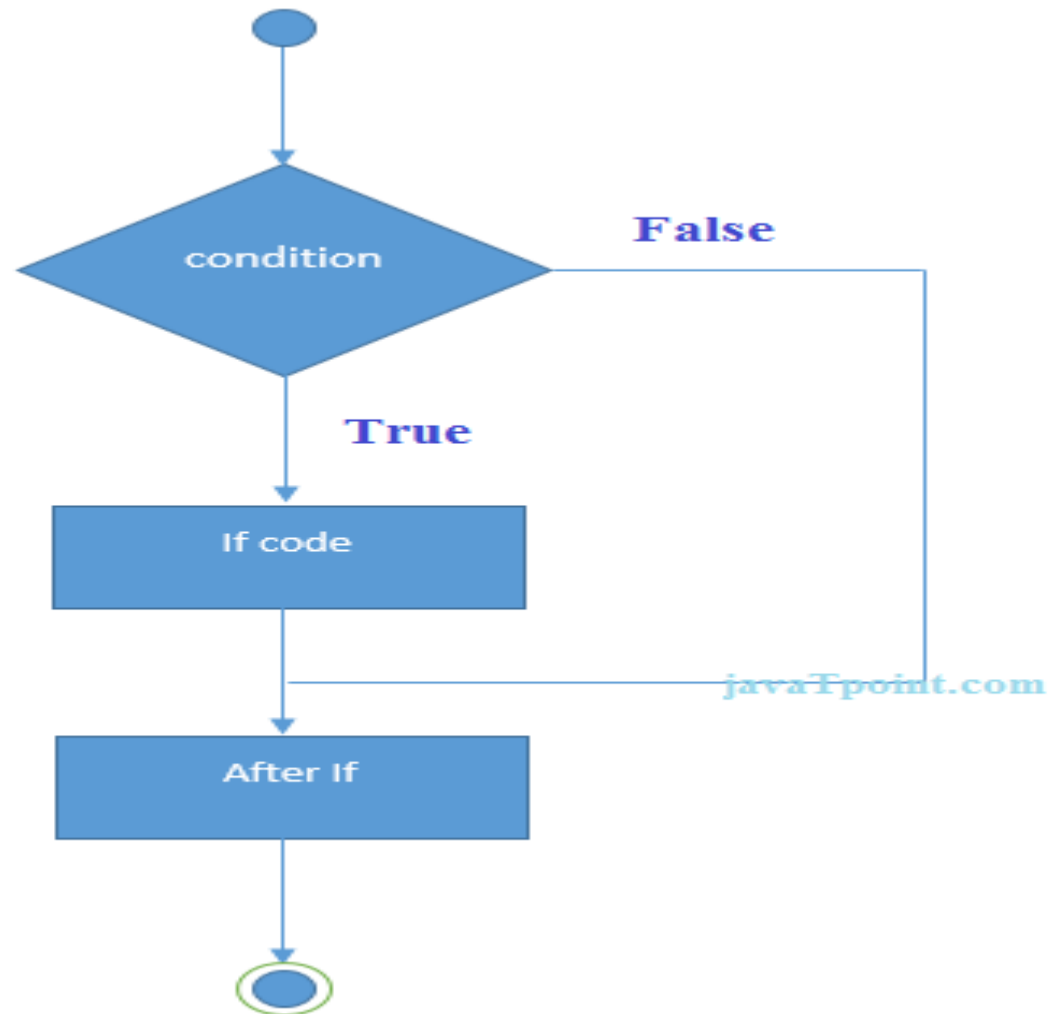
- If statement
- If-else statement
- If else-if ladder
- Nested if

If Statement

The if statement is used to check some given condition and perform some operations depending upon the correctness of that condition. It is mostly used in the scenario where we need to perform the different operations for the different conditions. The syntax of the if statement is given below.

```
if(expression){  
  //code to be executed  
}
```

Flowchart of if statement in C



Let's see a simple example of C language if statement.

NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH
DATA ANALYTICS--CP09

```
#include<stdio.h>
int main(){
int number=0;
printf("Enter a number:");
scanf("%d",&number);
if(number%2==0){
printf("%d is even number",number);
}
return 0;
}
```

Output

```
Enter a number:4
4 is even number
enter a number:5
```

Program to find the largest number of the three.

```
#include <stdio.h>
int main()
{
    int a, b, c;
    printf("Enter three numbers?");
```



```
scanf("%d %d %d",&a,&b,&c);
if(a>b && a>c)
{
    printf("%d is largest",a);
}
if(b>a && b > c)
{
    printf("%d is largest",b);
}
if(c>a && c>b)
{
    printf("%d is largest",c);
}
if(a == b && a == c)
{
    printf("All are equal");
}
}
```

Output

```
Enter three numbers?
12 23 34
```

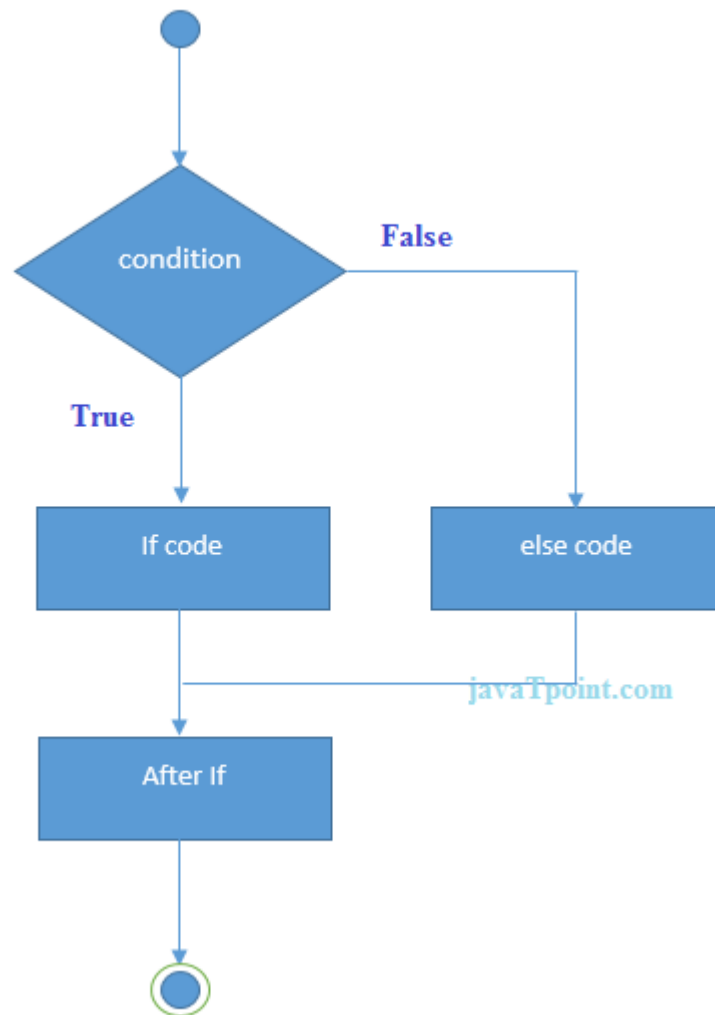
34 is largest

If-else Statement

The if-else statement is used to perform two operations for a single condition. The if-else statement is an extension to the if statement using which, we can perform two different operations, i.e., one is for the correctness of that condition, and the other is for the incorrectness of the condition. Here, we must notice that if and else block cannot be executed simultaneously. Using if-else statement is always preferable since it always invokes an otherwise case with every if condition. The syntax of the if-else statement is given below.

```
if(expression){  
    //code to be executed if condition is true  
}else{  
    //code to be executed if condition is false  
}
```

Flowchart of the if-else statement in C



NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH
DATA ANALYTICS--CP09

Let's see the simple example to check whether a number is even or odd using if-else statement in C language.

```
#include<stdio.h>
int main(){
int number=0;
printf("enter a number:");
scanf("%d",&number);
if(number%2==0){
printf("%d is even number",number);
}
else{
printf("%d is odd number",number);
}
return 0;
}
```

Output

```
enter a number:4
4 is even number
enter a number:5
5 is odd number
```

Program to check whether a person is eligible to vote or not.

```
#include <stdio.h>
int main()
{
    int age;
    printf("Enter your age?");
    scanf("%d",&age);
    if(age>=18)
    {
        printf("You are eligible to vote...");
    }
    else
    {
        printf("Sorry ... you can't vote");
    }
}
```

Output

```
Enter your age?18
You are eligible to vote...
Enter your age?13
Sorry ... you can't vote
```

If else-if ladder Statement

The if-else-if ladder statement is an extension to the if-else statement. It is used in the scenario where there are multiple cases to be performed for different conditions. In if-else-if ladder statement, if a condition is true then the statements defined in the if block will be executed, otherwise if some other condition is true then the statements defined in the else-if block will be executed, at the last if none of the condition is true then the statements defined in the else block will be executed. There are multiple else-if blocks possible. It is similar to the switch case statement where the default is executed instead of else block if none of the cases is matched.

```
if(condition1){  
    //code to be executed if condition1 is true  
}  
else if(condition2){  
    //code to be executed if condition2 is true  
}  
else if(condition3){  
    //code to be executed if condition3 is true  
}  
...  
else{
```

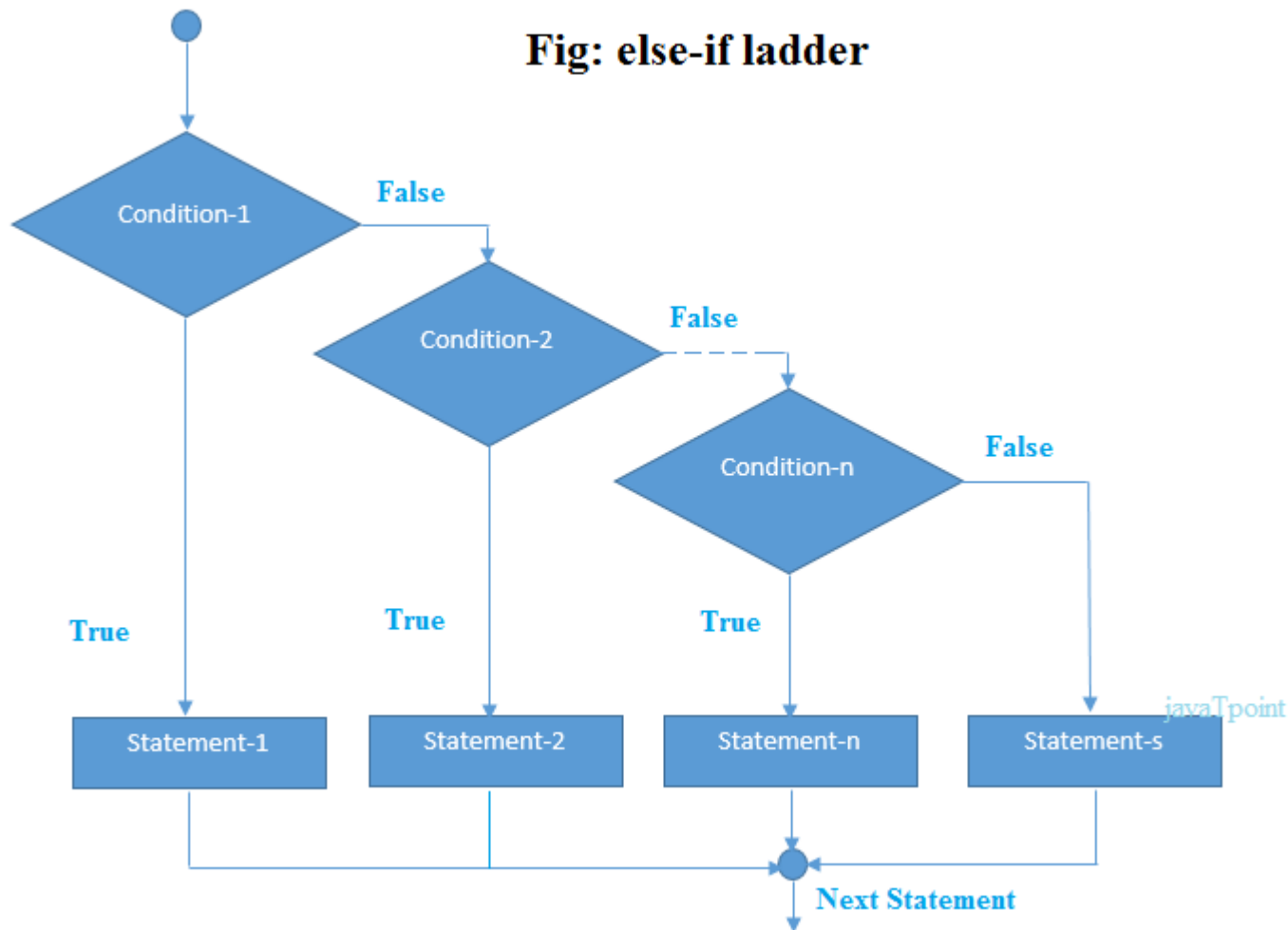
NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH DATA
ANALYTICS--CP09



```
//code to be executed if all the conditions are false  
}
```

Flowchart of else-if ladder statement in C

Fig: else-if ladder



The example of an if-else-if statement in C language is given below.

```
#include<stdio.h>
int main(){
int number=0;
printf("enter a number:");
scanf("%d",&number);
if(number==10){
printf("number is equals to 10");
}
else if(number==50){
printf("number is equal to 50");
}
else if(number==100){
printf("number is equal to 100");
}
else{
printf("number is not equal to 10, 50 or 100");
}
return 0;
}
```

Output

```
enter a number:4  
number is not equal to 10, 50 or 100  
enter a number:50  
number is equal to 50
```

Program to calculate the grade of the student according to the specified marks.

```
#include <stdio.h>  
int main()  
{  
    int marks;  
    printf("Enter your marks?");  
    scanf("%d",&marks);  
    if(marks > 85 && marks <= 100)  
    {  
        printf("Congrats ! you scored grade A ...");  
    }  
    else if (marks > 60 && marks <= 85)  
    {  
        printf("You scored grade B + ...");  
    }  
}
```

```
else if (marks > 40 && marks <= 60)
{
    printf("You scored grade B ...");
}
else if (marks > 30 && marks <= 40)
{
    printf("You scored grade C ...");
}
else
{
    printf("Sorry you are fail ...");
}
}
```

Output

```
Enter your marks?10
Sorry you are fail ...
Enter your marks?40
You scored grade C ...
Enter your marks?90
Congrats ! you scored grade A ...
```

C Switch Statement

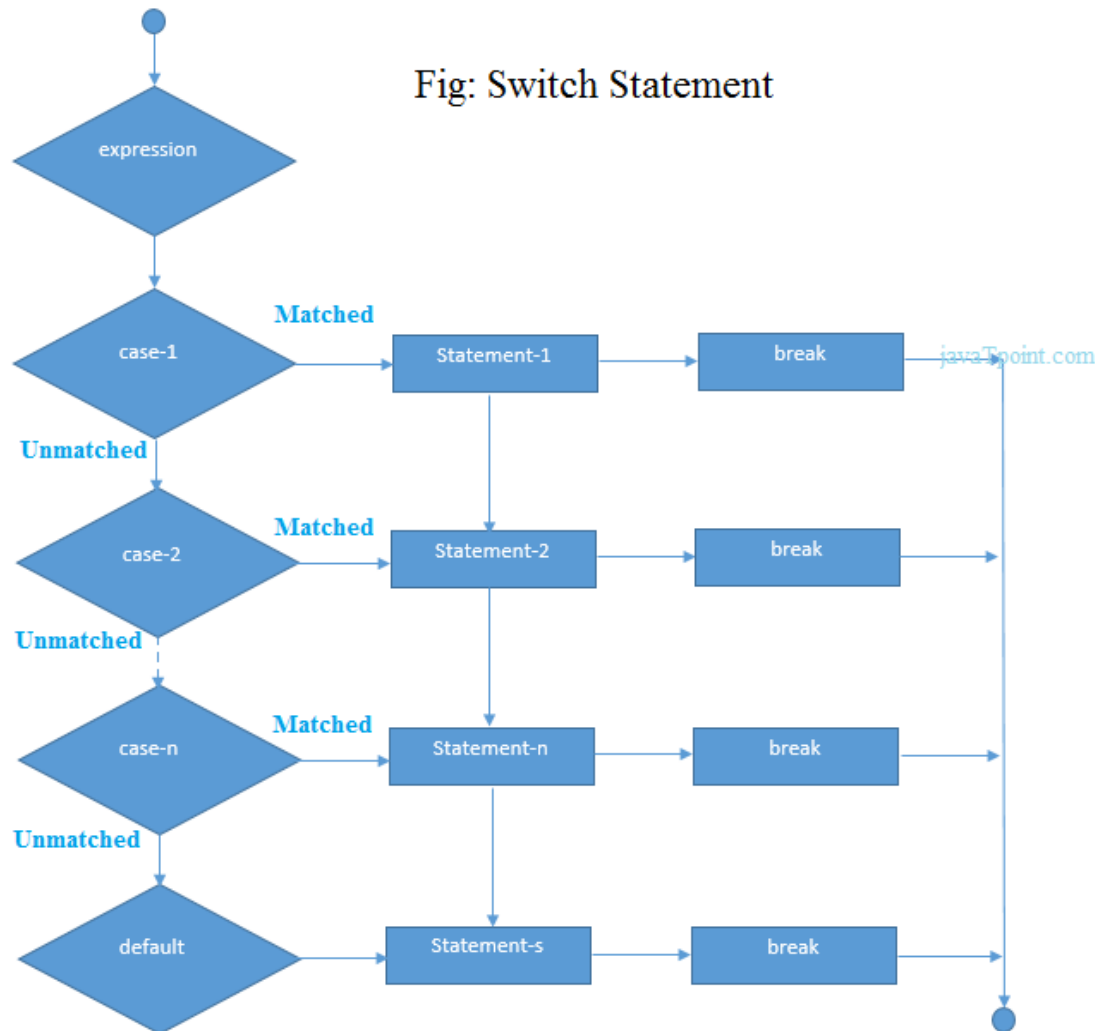
The switch statement in C is an alternate to if-else-if ladder statement which allows us to execute multiple operations for the different possible values of a single variable called switch variable. Here, We can define various statements in the multiple cases for the different values of a single variable.

The syntax of switch statement in c language is given below:

```
switch(expression){  
  case value1:  
    //code to be executed;  
    break; //optional  
  case value2:  
    //code to be executed;  
    break; //optional  
  .....  
  
  default:  
    code to be executed if all cases are not matched;
```

} **Flowchart of switch statement in C**

Fig: Switch Statement



Functioning of switch case statement

First, the integer expression specified in the switch statement is evaluated. This value is then matched one by one with the constant values given in the different cases. If a match is found, then all the statements specified in that case are executed along with the all the cases present after that case including the default statement. No two cases can have similar values. If the matched case contains a break statement, then all the cases present after that will be skipped, and the control comes out of the switch. Otherwise, all the cases following the matched case will be executed.

Let's see a simple example of c language switch statement.

```
#include<stdio.h>
int main(){
int number=0;
printf("enter a number:");
scanf("%d",&number);
switch(number){
case 10:
printf("number is equals to 10");
```

```
break;
case 50:
printf("number is equal to 50");
break;
case 100:
printf("number is equal to 100");
break;
default:
printf("number is not equal to 10, 50 or 100");
}
return 0;
}
```

Output

```
enter a number:4
number is not equal to 10, 50 or 100
enter a number:50
number is equal to 50
```

C Loops

The looping can be defined as repeating the same process multiple times until a specific condition satisfies. There are three types of loops used in the C language. In this part of the tutorial, we are going to learn all the aspects of C loops.

Why use loops in C language?

The looping simplifies the complex problems into the easy ones. It enables us to alter the flow of the program so that instead of writing the same code again and again, we can repeat the same code for a finite number of times. For example, if we need to print the first 10 natural numbers then, instead of using the printf statement 10 times, we can print inside a loop which runs up to 10 iterations.

Advantage of loops in C

- 1) It provides code reusability.
- 2) Using loops, we do not need to write the same code again and again.
- 2) Using loops, we can traverse over the elements of data structures (array or linked lists).

Types of C Loops

There are three types of loops in C language that is given below:

1. do while
2. while

3. for

do-while loop in C

The do-while loop continues until a given condition satisfies. It is also called post tested loop. It is used when it is necessary to execute the loop at least once (mostly menu driven programs).

The syntax of do-while loop in c language is given below:

```
do{  
    //code to be executed  
}while(condition);
```

Flowchart and Example of do-while loop

while loop in C

The while loop in c is to be used in the scenario where we don't know the number of iterations in advance. The block of statements is executed in the while loop until the condition specified in the while loop is satisfied. It is also called a pre-tested loop.

The syntax of while loop in c language is given below:

```
while(condition){  
    //code to be executed  
}
```

Flowchart and Example of while loop

for loop in C

The for loop is used in the case where we need to execute some part of the code until the given condition is satisfied. The for loop is also called as a per-tested loop. It is better to use for loop if the number of iteration is known in advance.

The syntax of for loop in c language is given below:

```
for(initialization;condition;incr/decr){  
    //code to be executed  
}
```

do while loop in C

The do while loop is a post tested loop. Using the do-while loop, we can repeat the execution of several parts of the statements. The do-while loop is mainly used in the case where we need to execute the loop at least once. The do-while loop is mostly

used in menu-driven programs where the termination condition depends upon the end user.

do while loop syntax

The syntax of the C language do-while loop is given below:

```
do{  
    //code to be executed  
}while(condition);
```

Example 1

```
#include<stdio.h>  
#include<stdlib.h>  
void main ()  
{  
    char c;  
    int choice,dummy;  
    do{  
        printf("\n1. Print Hello\n2. Print Javatpoint\n3. Exit\n");  
        scanf("%d",&choice);  
        switch(choice)  
        {  
            case 1 :
```

NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH
DATA ANALYTICS--CP09

```
printf("Hello");  
break;  
case 2:  
printf("Javatpoint");  
break;  
case 3:  
exit(0);  
break;  
default:  
printf("please enter valid choice");  
}  
printf("do you want to enter more?");  
scanf("%d",&dummy);  
scanf("%c",&c);  
}while(c=='y');  
}
```

Output

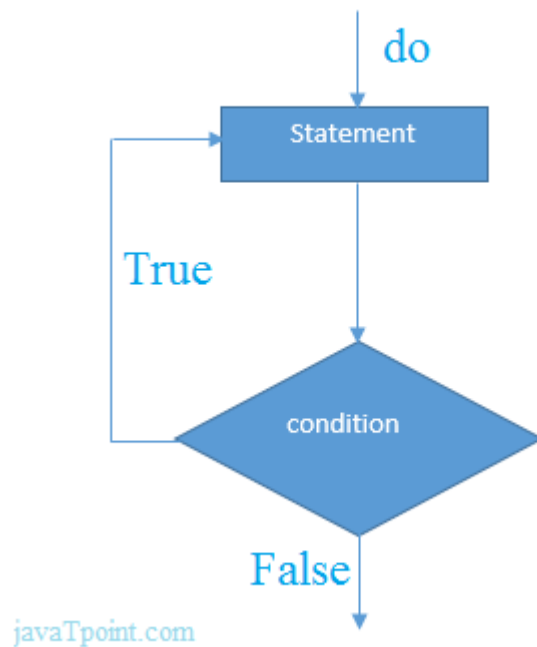
1. Print Hello
2. Print Javatpoint
3. Exit

```
1
Hello
do you want to enter more?
y
```

1. Print Hello
2. Print Javatpoint
3. Exit

```
2
Javatpoint
do you want to enter more?
n
```

Flowchart of do while loop



do while example

There is given the simple program of c language do while loop where we are printing the table of 1.

```
#include<stdio.h>
```

```
int main(){
    int i=1;
    do{
        printf("%d \n",i);
        i++;
    }while(i<=10);
    return 0;
}
```

Output

```
1
2
3
4
5
6
7
8
9
10
```

Program to print table for the given number using do while loop

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

NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH
DATA ANALYTICS--CP09

```
int i=1,number=0;
printf("Enter a number: ");
scanf("%d",&number);
do{
printf("%d \n",(number*i));
i++;
}while(i<=10);
return 0;
}
```

Output

Enter a number: 5

5
10
15
20
25
30
35
40
45

50

Enter a number: 10

10

20

30

40

50

60

70

80

90

100

Infinitive do while loop

The do-while loop will run infinite times if we pass any non-zero value as the conditional expression.

```
do{  
    //statement  
}while(1);
```

while loop in C

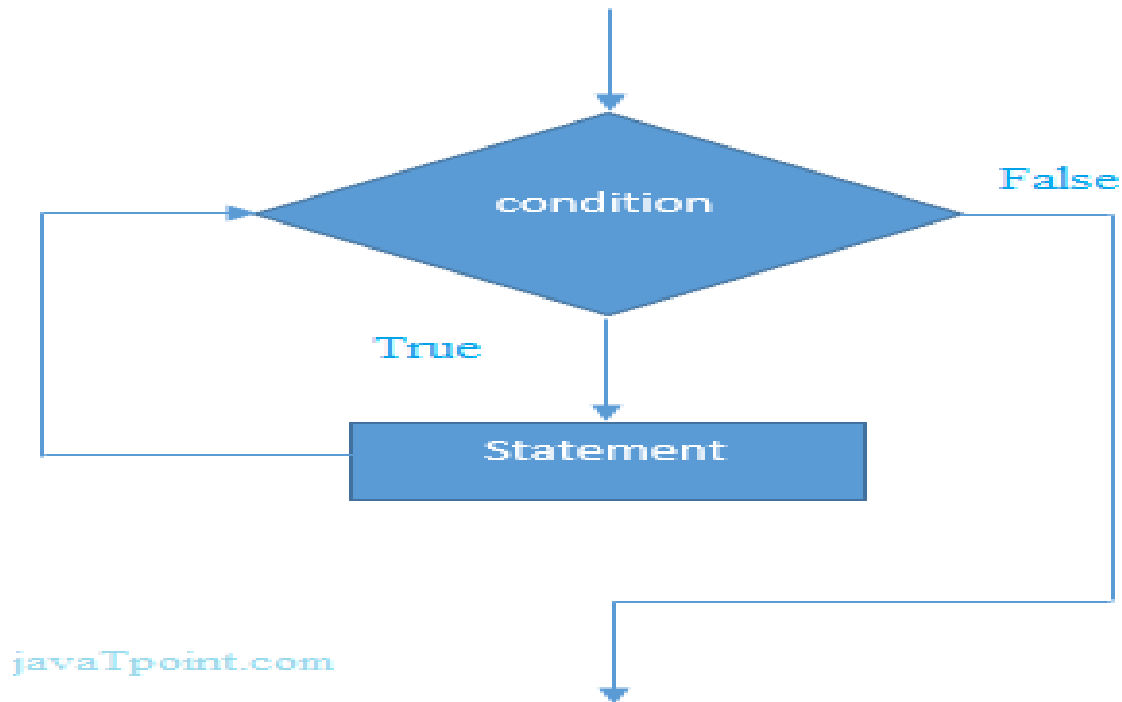
While loop is also known as a pre-tested loop. In general, a while loop allows a part of the code to be executed multiple times depending upon a given boolean condition. It can be viewed as a repeating if statement. The while loop is mostly used in the case where the number of iterations is not known in advance.

Syntax of while loop in C language

The syntax of while loop in c language is given below:

```
while(condition){  
    //code to be executed  
}
```

Flowchart of while loop in C



Example of the while loop in C language

Let's see the simple program of while loop that prints table of 1.

```
#include<stdio.h>
int main(){
int i=1;
while(i<=10){
printf("%d \n",i);
i++;
}
return 0;
}
```

Output

```
1
2
3
4
5
6
```

7
8
9
10

Program to print table for the given number using while loop in C

```
#include<stdio.h>
int main(){
    int i=1,number=0,b=9;
    printf("Enter a number: ");
    scanf("%d",&number);
    while(i<=10){
        printf("%d \n",(number*i));
        i++;
    }
    return 0;
}
```

Output

Enter a number: 50
50
100
150
200

NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH
DATA ANALYTICS--CP09

250

300

350

400

450

500

Enter a number: 100

100

200

300

400

500

600

700

800

900

1000

Properties of while loop

- A conditional expression is used to check the condition. The statements defined inside the while loop will repeatedly execute until the given condition fails.
- The condition will be true if it returns 0. The condition will be false if it returns any non-zero number.
- In while loop, the condition expression is compulsory.
- Running a while loop without a body is possible.
- We can have more than one conditional expression in while loop.
- If the loop body contains only one statement, then the braces are optional.

Example 1

```
#include<stdio.h>
void main ()
{
    int j = 1;
    while(j+=2,j<=10)
    {
        printf("%d ",j);
    }
    printf("%d",j);
}
```

}

Output

3 5 7 9 11

Example 2

```
#include<stdio.h>
void main ()
{
    while()
    {
        printf("hello Javatpoint");
    }
}
```

Output

compile time error: while loop can't be empty

Example 3

```
#include<stdio.h>
void main ()
{
    int x = 10, y = 2;
    while(x+y-1)
```



```
    {  
        printf("%d %d",x--,y--);  
    }  
}
```

Output

infinite loop

Infinitive while loop in C

If the expression passed in while loop results in any non-zero value then the loop will run the infinite number of times.

```
while(1){  
    //statement  
}
```

for loop in C

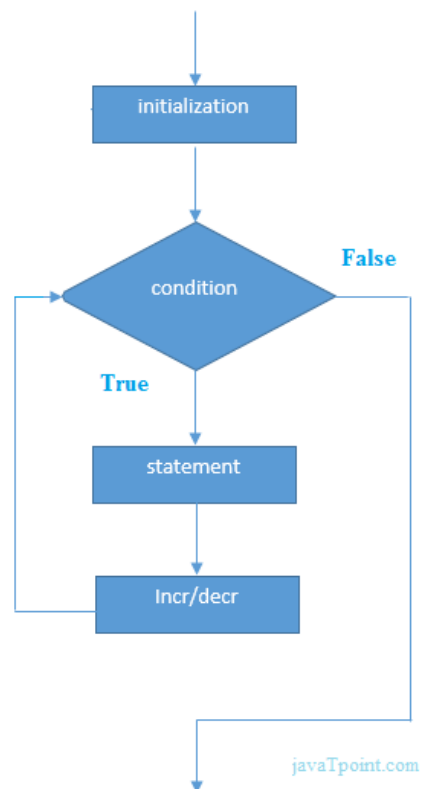
The **for loop in C language** is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like the array and linked list.

Syntax of for loop in C

The syntax of for loop in c language is given below:

```
for(Expression 1; Expression 2; Expression 3){  
    //code to be executed  
}
```

Flowchart of for loop in C



C for loop Examples

Let's see the simple program of for loop that prints table of 1.

```
#include<stdio.h>
int main(){
int i=0;
for(i=1;i<=10;i++){
printf("%d \n",i);
}
return 0;
}
```

Output

```
1
2
3
4
5
6
7
8
9
10
```

NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH
DATA ANALYTICS--CP09

C Program: Print table for the given number using C for loop

```
#include<stdio.h>
int main(){
int i=1,number=0;
printf("Enter a number: ");
scanf("%d",&number);
for(i=1;i<=10;i++){
printf("%d \n",(number*i));
}
return 0;
}
```

Output

Enter a number: 2

2
4
6
8
10
12
14

16

18

20

Enter a number: 1000

1000

2000

3000

4000

5000

6000

7000

8000

9000

10000

Properties of Expression 1

- The expression represents the initialization of the loop variable.
- We can initialize more than one variable in Expression 1.
- Expression 1 is optional.
- In C, we can not declare the variables in Expression 1. However, It can be an exception in some compilers.

Example 1

```
#include <stdio.h>
int main()
{
    int a,b,c;
    for(a=0,b=12,c=23;a<2;a++)
    {
        printf("%d ",a+b+c);
    }
}
```

Output

35 36

Example 2

```
#include <stdio.h>
int main()
{
    int i=1;
```

```
for(;i<5;i++)  
{  
    printf("%d ",i);  
}  
}
```

Output

1 2 3 4

Properties of Expression 2

- Expression 2 is a conditional expression. It checks for a specific condition to be satisfied. If it is not, the loop is terminated.
- Expression 2 can have more than one condition. However, the loop will iterate until the last condition becomes false. Other conditions will be treated as statements.
- Expression 2 is optional.
- Expression 2 can perform the task of expression 1 and expression 3. That is, we can initialize the variable as well as update the loop variable in expression 2 itself.
- We can pass zero or non-zero value in expression 2. However, in C, any non-zero value is true, and zero is false by default.

Example 1

```
#include <stdio.h>
int main()
{
    int i;
    for(i=0;i<=4;i++)
    {
        printf("%d ",i);
    }
}
```

output

0 1 2 3 4

Example 2

```
#include <stdio.h>
int main()
{
    int i,j,k;
```



```
for(i=0,j=0,k=0;i<4,k<8,j<10;i++)
{
    printf("%d %d %d\n",i,j,k);
    j+=2;
    k+=3;
}
```

Output

```
0 0 0
1 2 3
2 4 6
3 6 9
4 8 12
```

Example 3

```
#include <stdio.h>
int main()
{
    int i;
    for(i=0;;i++)
```

```
    {  
        printf("%d",i);  
    }  
}
```

Output

infinite loop

Properties of Expression 3

- Expression 3 is used to update the loop variable.
- We can update more than one variable at the same time.
- Expression 3 is optional.

Example 1

```
#include<stdio.h>  
void main ()  
{  
    int i=0,j=2;  
    for(i = 0;i<5;i++,j=j+2)  
    {
```

```
        printf("%d %d\n",i,j);  
    }  
}
```

Output

```
0 2  
1 4  
2 6  
3 8  
4 10
```

Loop body

The braces { } are used to define the scope of the loop. However, if the loop contains only one statement, then we don't need to use braces. A loop without a body is possible. The braces work as a block separator, i.e., the value variable declared inside for loop is valid only for that block and not outside. Consider the following example.

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

```
for(i=0;i<10;i++)  
{  
    int i = 20;  
    printf("%d ",i);  
}  
}
```

Output

20 20 20 20 20 20 20 20 20 20

C break statement

The break is a keyword in C which is used to bring the program control out of the loop. The break statement is used inside loops or switch statement. The break statement breaks the loop one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops. The break statement in C can be used in the following two scenarios:

1. With switch case
2. With loop

Syntax:

//loop or switch case

break;

Flowchart of break in c

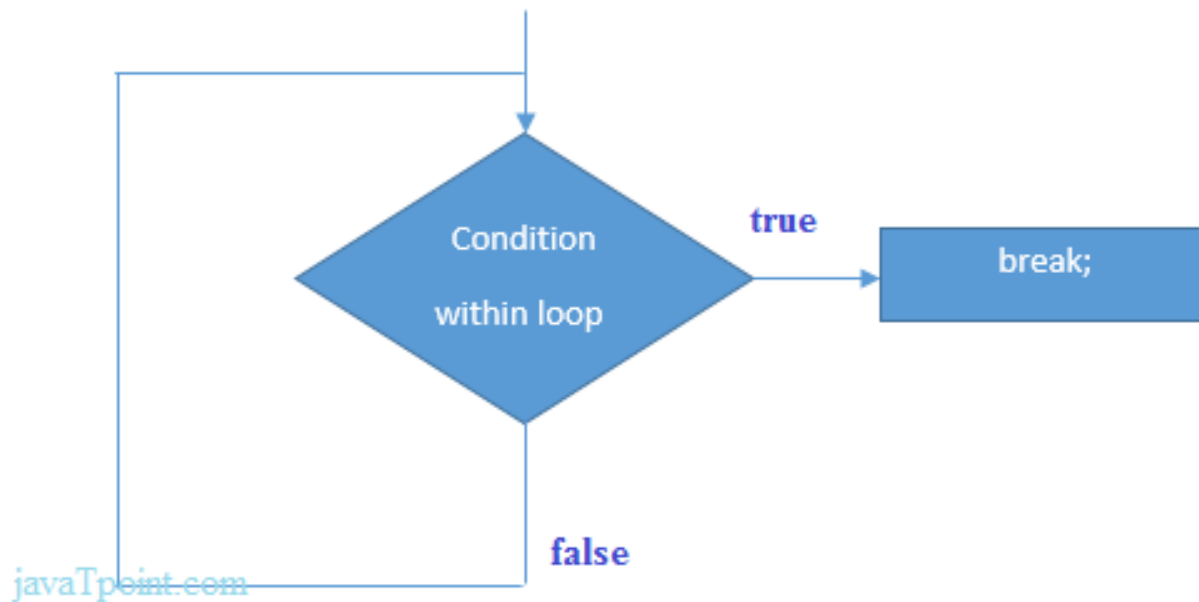


Figure: Flowchart of break statement

Example

```
#include<stdio.h>
#include<stdlib.h>
void main ()
{
    int i;
    for(i = 0; i<10; i++)
    {
        printf("%d ",i);
        if(i == 5)
            break;
    }
    printf("came outside of loop i = %d",i);
}
```

Output

0 1 2 3 4 5 came outside of loop i = 5

A function is a group of statements that together perform a task. Every C program has at least one function, which is **main()**, and all the most trivial programs can define additional functions.

You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division is such that each function performs a specific task.

A function **declaration** tells the compiler about a function's name, return type, and parameters. A function **definition** provides the actual body of the function.

The C standard library provides numerous built-in functions that your program can call. For example, **strcat()** to concatenate two strings, **memcpy()** to copy one memory location to another location, and many more functions.

A function can also be referred as a method or a sub-routine or a procedure, etc.

Defining a Function

The general form of a function definition in C programming language is as follows –

```
return_type function_name( parameter list ) {  
    body of the function  
}
```

A function definition in C programming consists of a *function header* and a *function body*. Here are all the parts of a function –

NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH
DATA ANALYTICS--CP09

- **Return Type** – A function may return a value. The **return_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the **return_type** is the keyword **void**.
- **Function Name** – This is the actual name of the function. The function name and the parameter list together constitute the function signature.
- **Parameters** – A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
- **Function Body** – The function body contains a collection of statements that define what the function does.

Example

Given below is the source code for a function called **max()**. This function takes two parameters num1 and num2 and returns the maximum value between the two –

```
/* function returning the max between two numbers */  
int max(int num1, int num2) {  
  
    /* local variable declaration */  
    int result;  
  
    if (num1 > num2)  
        result = num1;  
    else
```

```
result = num2;  
  
return result;  
}
```

Function Declarations

A function **declaration** tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

A function declaration has the following parts –

```
return_type function_name( parameter list );
```

For the above defined function max(), the function declaration is as follows –

```
int max(int num1, int num2);
```

Parameter names are not important in function declaration only their type is required, so the following is also a valid declaration –

```
int max(int, int);
```

Function declaration is required when you define a function in one source file and you call that function in another file. In such case, you should declare the function at the top of the file calling the function.

Calling a Function

While creating a C function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task.

When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.

To call a function, you simply need to pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value. For example –

```
#include <stdio.h>

/* function declaration */

int max(int num1, int num2);
```

NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH
DATA ANALYTICS--CP09

```
int main () {  
  
    /* local variable definition */  
  
    int a = 100;  
    int b = 200;  
    int ret;  
  
    /* calling a function to get max value */  
    ret = max(a, b);  
  
    printf( "Max value is : %d\n", ret );  
  
    return 0;
```

```
}

/* function returning the max between two numbers */
int max(int num1, int num2) {

    /* local variable declaration */
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
```

```
}
```

We have kept max() along with main() and compiled the source code. While running the final executable, it would produce the following result –

Max value is : 200

Function Arguments

If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the **formal parameters** of the function.

Formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit.

While calling a function, there are two ways in which arguments can be passed to a function –

Sr.No.	Call Type & Description
1	<p><u>Call by value</u></p> <p>This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.</p>
2	<p><u>Call by reference</u></p> <p>This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.</p>

By default, C uses **call by value** to pass arguments. In general, it means the code within a function cannot alter the arguments used to call the function.

6 Problem solving

Introduction to Computational Thinking

Some quotes about Computational Thinking

“Computational Thinking is the new literacy of the 21st century. The goal is for it to be a fundamental skill used by everyone in the world by the middle of the 21st century. Just like reading, writing and arithmetic.”

“Even people who aren't going into computer science or engineering programs, should be exposed to computer science. Programming teaches you how to take problems, break them down into smaller problems, and solve them. Any kind of job that involves problem solving can benefit from computing. ”

“Computers are extremely talented at performing menial tasks quickly; Computational Thinkers can leverage that power to find solutions to previously impossible problems.”

“I think everyone should get a little exposure to computer science because it really forces you to think in a slightly different way, and it's a skill that you can apply in life in general.”

5.4 What is computational thinking?

Who needs it?

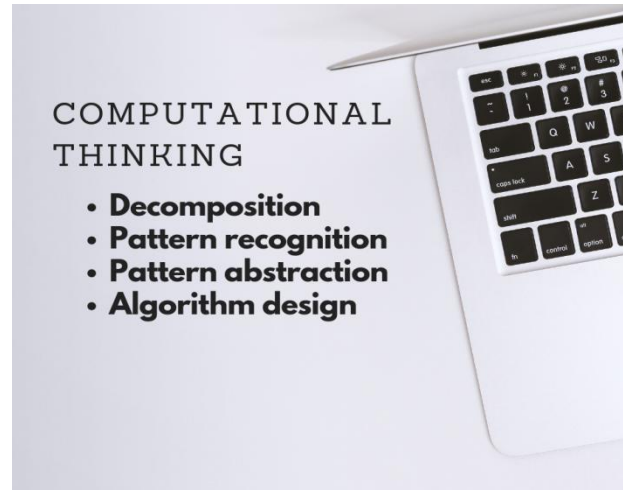
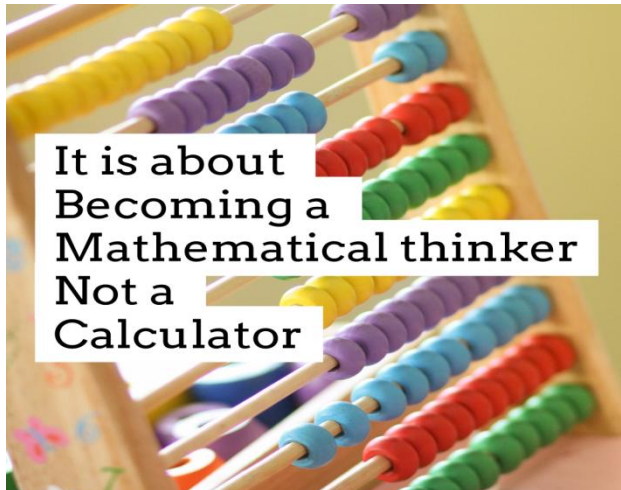
Why?

What is Computational Thinking?

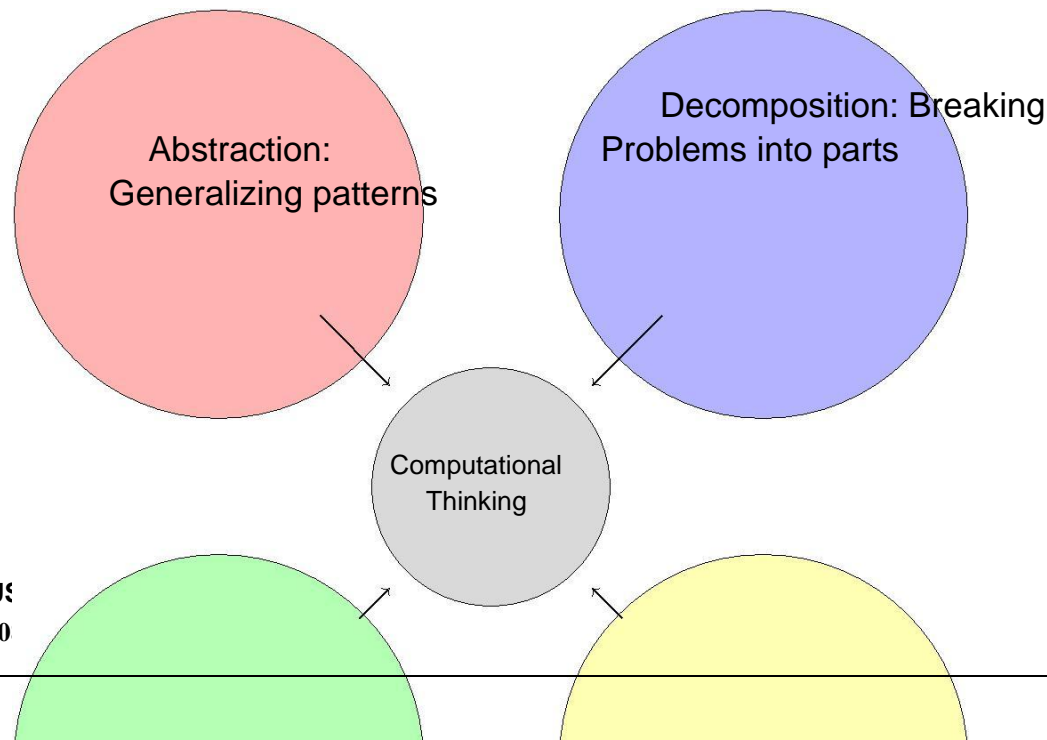
Computational thinking is the step that comes before programming. It's the process of breaking down a problem into simple enough steps that even a computer would understand. We all know that computers take instructions very literally, sometimes to comic results. If we don't provide computers with instructions that are precise and detailed, your algorithm might forget vital actions that most people take for granted.

For example, consider a simple activity like brushing your teeth. At first it sounds like a simple enough task, but in fact, brushing your teeth involves many simple steps. First, you'll need a toothbrush and toothpaste. You'll need a sink with cold water. You'll need to put the toothpaste on the brush. Don't forget to turn on the water and run your brush underneath. As you see, such a simple activity actually involves many steps, if you miss one step or put one out of order you might end up with a huge mess!

NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH
DATA ANALYTICS--CP09



Computational Thinking is a problem-solving process that includes the following characteristics.



Pattern Recognition:

identifying patterns

and trends

Algorithms: Designing

& implementing methods

to solve problems

5.6 Developing algorithms

What is an algorithm?

“An algorithm is like a recipe, it must be followed exactly, it must be unambiguous, and it must have an end.”

An algorithm is a precise sequence of steps to solve a problem.

An algorithm should generate a solution. The solution may NOT be the \best" solution to the problem. The algorithm may NOT be the most efficient way to solve the problem.

NETTUR TECHNICAL TRAINING FOUNDATION
COMMON SYLLABUS FOR DIPLOMA IN INFORMATION TECHNOLOGY WITH
DATA ANALYTICS--CP09

An issue that arises in designing an algorithm is that we have to describe the algorithm in an orderly and unambiguous way. This is not as easy as it may seem.

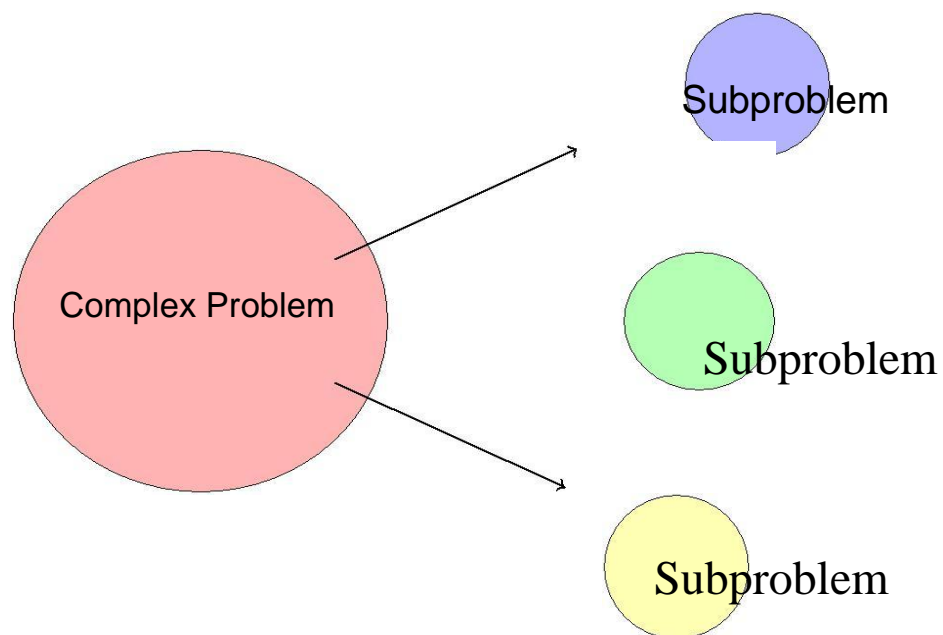
As an example, consider the following video from an introductory computer science course at Harvard where the class is attempting to give

three volunteers an algorithm for making a peanut butter and jelly sandwich. Note that the center participant has a different type of jelly container than the other two. This is to illustrate that we often have to take care of special cases when designing an algorithm.

5.7 Decomposition: Breaking problems into simple parts



“If you can't solve a problem, then there is an easier problem you can solve: find it”



5.8 Abstraction

1) Abstraction is the process of filtering out unnecessary information.

2) Abstraction allows us to create a general idea of what the problem is and how to solve it.

Example: Abstracting a general procedure for baking cookies



1. Bring butter and eggs to room temperature
2. Sift dry ingredients
3. Cream sugars and butter
4. Add eggs and vanilla
5. Mix in dry ingredients
6. Mix in add-ins like chips, nuts, etc.
7. Bake

5.7 Decomposition

Decomposition is breaking down complex problems into smaller, more manageable chunks. With young children, you can teach decomposition by getting them to teach you how to perform a simple task. Any simple activity like brushing teeth, making breakfast, or reading a book will work. Students will need to break down the task into small simple steps. Be sure to give them a challenge and only do what is asked of you! With this activity, kids will quickly see how important it is to give the EXACT instructions.

Decomposition allows students to assess the problem at hand and figure out all of the steps needed to make the task happen. One way to teach older students the skill of decomposition is to have them build something by only showing them the finished project. Give them the supplies needed, and get them to make it without instructions. Students will need to figure out the steps needed to complete the final project.

Decomposition is an important life skill in the future when students and adults need to take on larger tasks. Students will learn ways to delegate in group projects and build time management skills.

5.9 Pattern recognition

Pattern recognition is simply looking for patterns in the puzzles and determining could any of the problems or solutions we've

CP09

encountered in the past apply here? What have we learned in the past that may help us sort out this problem? If you've ever built a piece of IKEA furniture, you'll understand the importance of patterns. When building an IKEA drawer unit it will likely take you much longer to assemble the first drawer than the fourth or fifth. When we repeat steps in our build we learn how to solve the instructions more quickly and learn from our mistakes. The painstaking process of assembling that first part teaches us the skills to perform the process more efficiently in the future.



5.10 Designing algorithms

- _ Determine the right steps
- _ Determine the correct order of the steps
- _ Follow the steps completely
- _ Verify that the algorithm is working correctly
- _ Determine if algorithm is the "best" approach for solving problem

Example. Computer algorithm for multiplying a number by an integer using only addition

Assume that the computer only has the ability to do addition and we

want to multiply something like

351:6 _ 14

Let a be a given number which we want to multiply by an integer b

product = 0

for i = 1, b

product = product + a

end

This is called **pseudo code** for our algorithm because it does not use

language specific terms.

5.11 Computer Hardware vs Computer Software

Computer hardware is any physical device used in or with your computer.

This includes your monitor, mouse, keyboard, hard drive, sound card, video card, etc.

Computer software is a general term used to describe a collection of

computer programs that perform some task on a computer system.

Computer software is generally divided into two classes { system soft-

ware and application software. Operating systems, utilities, device drivers, programming languages, interpreters are examples of system

software. Application software are a group of computer software that

are created to help the user complete tasks such as creating documents,

performing calculations, storing and managing data, playing videos, etc.

This includes programs like Microsoft Word, Adobe Acrobat, etc.

Computer hardware operates under the control of software.

We are interested in the [algorithms](#) that are contained within the soft-ware.

What is an algorithm?

“An algorithm is like a recipe, it must be followed exactly, it must be unambiguous, and it must have an end.”

- The word *algorithm* comes from the name of a Persian mathematician Abu Ja’far Mohammed ibn-i Musa al Khowarizmi.
- In computer science, this word refers to a special method useable by a computer for solution of a problem. The statement of the problem specifies in general terms the desired input/output relationship.
- For example, sorting a given sequence of numbers into nondecreasing order provides fertile ground for introducing many standard design techniques and analysis tools.

An algorithm is a precise sequence of steps to solve a problem.

An algorithm should generate a solution. The solution may NOT be the “best” solution to the problem. The algorithm may NOT be the most efficient way to solve the problem.

An issue that arises in designing an algorithm is that we have to describe the algorithm in an orderly and unambiguous way. This is not as easy as it may seem.

As an example, consider the following video from an introductory computer science course at Harvard where the class is attempting to give three volunteers an algorithm for making a peanut butter and jelly sandwich. Note that the center participant has a different type of jelly container than the other two. This is to illustrate that we often have to take care of special cases when designing an algorithm.

No Man's Sky: A Game Crafted by Algorithms

The virtual universe in this program is generated by an algorithm, not by an artist's pen. The generated universe is, of course, not infinite but “if you were to visit one virtual planet every second then our own sun will have died before you have seen them all.”

The most common example of this type of game is Minecraft which creates a unique world for each of its players, randomly arranging rocks and lakes from a limited palette of bricks whenever someone begins a new game. But No Man's Sky is far more complex and sophisticated because the algorithm contains rules for the physics, biology, etc. of each virtually generated planet. The tens of millions of planets that comprise the universe are all unique. Each is generated when a player discovers it, and is subject to the laws of its respective solar systems and vulnerable to natural erosion. The multitude of creatures that inhabit the universe dynamically breed and genetically mutate as time progresses.



The problem of sorting

Input: sequence $\langle a_1, a_2, \dots, a_n \rangle$ of numbers.

Output: permutation $\langle a'_1, a'_2, \dots, a'_n \rangle$ such that $a'_1 \leq a'_2 \leq \dots \leq a'_n$.

Example:

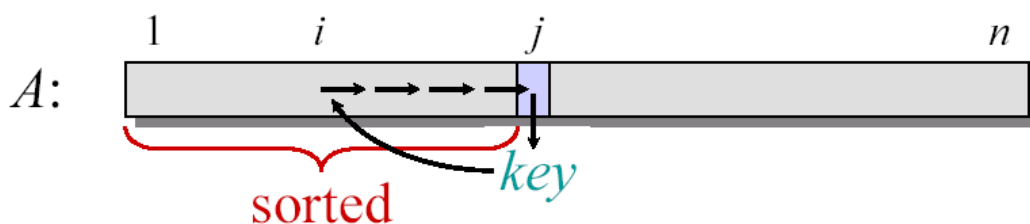
Input: 8 2 4 9 3 6

Output: 2 3 4 6 8 9

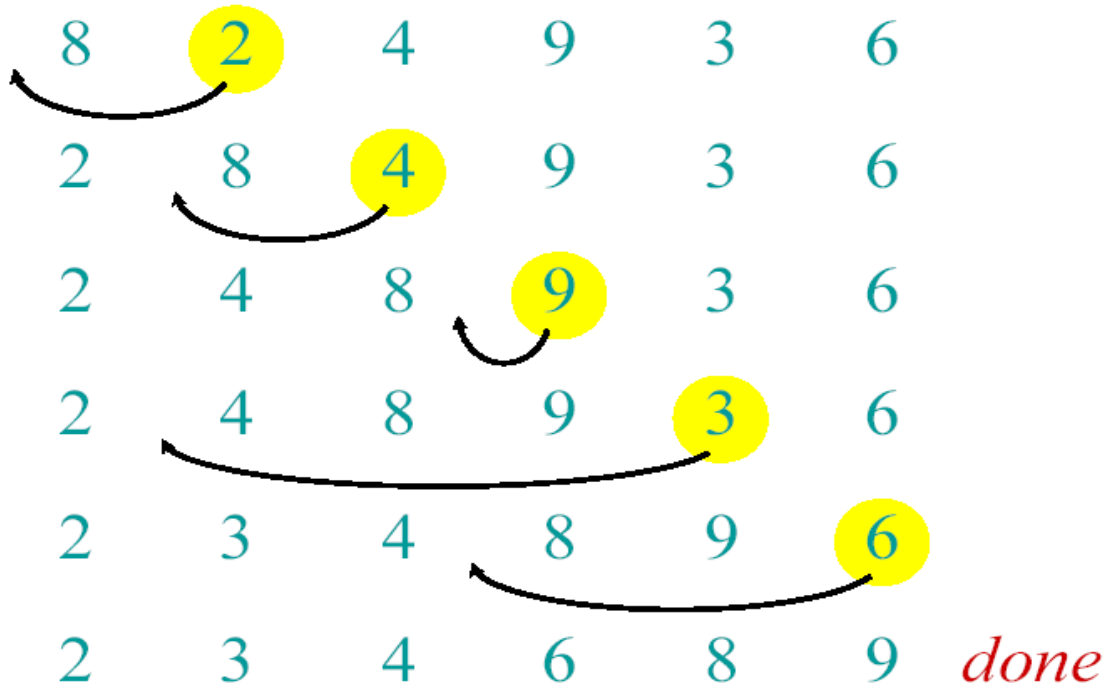
Insertion Sort

“pseudocode” {

```
INSERTION-SORT ( $A, n$ )  ▷  $A[1 \dots n]$ 
  for  $j \leftarrow 2$  to  $n$ 
    do  $key \leftarrow A[j]$ 
       $i \leftarrow j - 1$ 
      while  $i > 0$  and  $A[i] > key$ 
        do  $A[i+1] \leftarrow A[i]$ 
           $i \leftarrow i - 1$ 
       $A[i+1] = key$ 
```



Example of Insertion Sort



Discuss till more algorithms:

Analysis of algorithms

The theoretical study of computer-program performance and resource usage.

What's more important than performance?

- modularity
- correctness
- maintainability
- functionality
- robustness
- user-friendliness
- programmer time

- simplicity
- extensibility
- reliability

Why study algorithms and performance?

- Algorithms help us to understand scalability.
- Performance often draws the line between what is feasible and what is impossible.
- Algorithmic mathematics provides a *language* for talking about program behavior.
- The lessons of program performance generalize to other computing resources.
- Speed is fun!

Running Time

The running time depends on the input: an already sorted sequence is easier to sort.

- Parameterize the running time by the size of the input, since short sequences are easier to sort than long ones.
- Generally, we seek upper bounds on the running time, because everybody likes a guarantee.

5.12 Final Project:Applying Computational Thinking

Case study make trainees into some number of groups

Issues in Computational Thinking

What problems can computers help us with? What is different about

how a computer seeks a solution? Why are computer solutions oftennot quite the same as what a human would have found?

CP09

How do computer programmers write out instructions that tell a computer how to solve a problem, what problem to solve, and how to report the answer?

How does a computer \think"? How does it receive instructions, store information in a memory, represent and manipulate the numbers and words and pictures and ideas that we refer to as we think about a problem?