Certificate in Computer Applications (CCA)
Study Material

CCA -101: Fundamentals of IT & Programming
Part 4 (Unit 4.1 - Unit 4.5)

Supported by
Institute of Management Studies (IMS),
Ghaziabad-UP

About CCA Program

The certificate program focuses on computer fundamentals. This program provides a comprehensive introduction to Fundamentals of Information Technology; Computer Applications; Internet & Communication Technologies; Web Programming; and Soft Skills.

The program is designed and conducted by CSC Academy along with one of the leading Management Institute, Institute of Management Studies, Ghaziabad (UP). Some of the core subject faculty are associated in delivering this program.

After the completion of this course, student will be able to:

- Get a basic understanding of personal computers and their operations.
- Use of MS Office Tools - Like MS word, MS excel and Power point presentations
- Understand basics of Programming.
- Recognize and describe the working of Computer Networks.
- Get familiar with the basics of communication skills
- Develop good skills at writing business letters, emails, minutes of meeting and other business correspondence.
- Design and Implement interactive, responsive web site using HTM5L, CSS5 and JavaScript.
- Build Dynamic web site using server-side PHP Programming and Database connectivity.

The CCA program covers five course modules:

Unit 101: Fundamentals of IT & Programming

Unit 102: Data Communications

Unit 103: Soft Skills & Communications

Unit 104: Web Technologies

Unit 105: Cyber Security

The objective of this study material is to provide the students to enable them to obtain knowledge and skills in the related subject. This material is not in itself to be read alone, and student should use this in addition to the CCA online e-learning content study. In case students need any further clarifications or have any suggestions to make for further improvement of the material contained herein, they may give the same at CSC Academy Centre.

All care has been taken to provide content in a manner useful to the students.

Permission of the CSC Academy is essential for reproduction of any portion of this material.

Edition : June 2020

Published by : CSC Academy

About CSC Academy

CSC Academy was setup in 2017 that provides access to professional learning for learners of diverse backgrounds and educational needs. The CSC Academy is a not-for-profit society under the Societies Registration Act 1860 (Act 21 of 1860), as applicable to the Union of Delhi with its registered office in Delhi. The CSC Academy board comprises of the Additional Secretary, Ministry of Electronics & Information Technology, Government of India as Chairman, and others reputed members from academia. CSC Academy has received certificate from Income Tax Department under section 12 AA and 80 G.

The CSC Academy is committed to teaching, delivering of specialized courses/ training programs, leadership, communication skills and promotion of entrepreneurship among the rural masses in India. Presently, the CSC Academy is delivering various Government of India sponsored skill and education programs, in addition to courses from private sector.

About Institute of Management Studies, Ghaziabad (UP)

IMS Ghaziabad is a pioneer institute for management education in Northern India. It is the first institute of IMS Society Ghaziabad with 30 glorious years of excellence. IMS Ghaziabad offers full time AICTE approved & NBA accredited PGDM Programme which has been awarded the MBA equivalent status by the Association of Indian Universities (AIU), PGDM - International Business, PGDM - Big Data Analytics and MCA Programme are approved by AICTE and affiliated to Dr APJ AKTU, Lucknow.

Since its foundation IMS Ghaziabad has gathered a lot of feathers in its cap with global accreditations and memberships such as Accreditation Services for International Colleges (U.K), AACSB Business Education Alliance, National Assessment and Accreditation Council - 'A' Grade.

IMS Ghaziabad is amongst Top 10 best B-Schools in North India as per latest MBA and B School Rankings. It has been awarded as the "Best Campus for Industry Oriented Management Education in India / Asia Pacific 2019" by ASSOCHAM and the Education Post. It has been ranked as 5th in North India and 15th in India by Times of India B School Survey, February 2019, A++ Institute in Delhi - NCR by 9th Chronicle B-School Survey 2018.

Table of Contents

Course Outline

Course Objective

This subject aims to introduce skills relating to basic concepts and terminology of information technology & programming.

Course Outcomes

At the end of this course, student should be able to:

1. Understand basic concepts of I.T.
2. Have a basic understanding of personal computers and their operations.
3. Able to use MS office tools.
4. Understand basics of Programming.

Course Details

Unit I Introduction

Introduction to computers: definitions, evolution, characteristics, Organization of a Computer, Classifications, Distributed Computers, Parallel Computers.

Computer Memory: Random Access Memory (RAM), Read Only Memory (ROM), External Memory (Secondary Memory), Compact Disk Read Only Memory, Magnetic Storage Drives, USB.

Software: Types of S/W - System Software: Operating System, Utility Programs Application Software, Overview of proprietary software, Overview of open source technology.

UNIT II Introduction to MS Word

MS Word Processing basics: Menu Bar, Using the Icons below Menu Bar; Opening and closing Documents: Save and Save as, Page Setup, Print Preview.

Text Creation and manipulation: Document Creation, Editing Text, Text Selection, Cut, Copy and Paste, Spell check.

Formatting the Text: Font and Size selection, Alignment of Text, Paragraph Indenting, Bullets and Numbering, Changing case;

Formatting a document: Set page margin, paragraphs and sections within a document, Adjust indents and hanging indents;

Table Manipulation: Draw Table, Changing cell width and height, Alignment of Text in cell, Delete / Insertion of row and column Border and shading, Table Formula.

UNIT III Spreadsheets and Presentations

Spread Sheet: Opening of Spread Sheet, Addressing of Cells, Printing of Spread Sheet, Saving Workbooks.

Manipulation of Cells: Entering Text, Numbers and Dates, Creating Text, Number and Date Series, Editing Worksheet Data, Inserting and Deleting Rows, Column, Changing Cell Height and Width.

Formulas and Function: Using Formulas, Function, basic mathematical operators, using AutoSum etc., using formulas with multiple cell references;

Presentation - Basic Concepts of presentation: Using PowerPoint, Opening A Power Point Presentation, Saving A Presentation; Creation of Presentation using a Template, Creating a Blank Presentation, Entering and Editing Text, Inserting and Deleting Slides in a Presentation; **Preparation of Slides:** Inserting Word Table or An Excel Worksheet, Inserting Other Objects,

UNIT IV Introduction to Programming

Programming Language: Machine Language, Assembly Language, High Level Language their advantages & disadvantages. Basic concepts – data types and its representation in programming, basic arithmetic operations – addition, multiplication, division, modulus; conditional checks, relational and comparisons and loops

Reference books

1. Introduction to Information Systems,JamesO'Brien,GeorgeMarakas,TMH
2. “Information Technology for Management”, (2010) Behl, Ramesh, 1st Ed Tata McGraw Hill, NewDelhi
3. Alexis & Mathews: “Fundamentals of Information Technology”, VikasPublication.
4. Turban - Information technology for Management : Transforming Organization in Digital Economy 7/e-Wiley
5. Henry Lucas, Information Technology For Management,TMH

Unit 4 Programming Languages

Unit 4.1: Introduction to Programming Languages

Machine Language

- A computer programming language consisting of binary instructions which a computer can respond to directly.
- Sometimes, it is referred to as **machine** code or object code. **machine language** is a collection of binary digits or bits that the computer reads

- A computer cannot directly understand the **programming languages** used to create computer programs, so the program code must be compiled. Example: 01001000, 01100101, 01101100, 01101100 etc

Advantages:

- This language makes fast and efficient use of the computer.
- It requires no translator to translate the code. It is directly understood by the computer.

Disadvantages:

- All memory addresses have to be remembered.
- All operation codes have to be remembered.

Assembly Language

- An assembly language is a low-level programming language in which there is a very strong correspondence between the program's statements and the architecture's machine code instructions.
- A program written in assembly language consists of a series of mnemonic processor instructions and meta-statements (known as directives, pseudo-instructions). Assembly language instructions usually consist of an opcode mnemonic followed by a list of data, arguments or parameters.

- These are translated by an assembler into machine language instructions that can be loaded into memory and executed.
- Example: MOV AL 61h ;(Meaning – Load AL with 61 hex, MOV is abbreviation of Move)

Advantages Of Assembly Language

- Programs written in machine language are replaceable by mnemonics which are easier to remember.
- Memory Efficient.
- It is not required to keep track of memory locations.
- Faster in speed.
- Easy to make insertions and deletions.
- Hardware Oriented.
- Requires fewer instructions to accomplish the same result.

Disadvantages Of Assembly Language

- Long programs written in such languages cannot be executed on small sized computers.
- It takes lot of time to code or write the program, as it is more complex in nature.
- Difficult to remember the syntax.
- Lack of portability of program between computers of different makes.

High Level Language

- A high-level language is a programming language that enables development of a program in a much more user-friendly programming context.
- This language is a programming_language with strong abstraction about the details of the computer in contrast to low-level programming language (Assembly Language).
- Ex: C, C++, Java

- High level languages are grouped in two categories based on execution model – **compiled** or **interpreted** languages.
- Compiler and interpreter are used to convert the high level language into machine level language. The program written in high level language is known as source program and the corresponding machine level language program is called as object program.

- **Compiler** read the program at-a-time and searches the error and lists them. If the program is error free then it is converted into object program.
- When program size is large then compiler is preferred. Whereas **interpreter** read only one line of the source code and convert it to object code.

Advantages of High level language

- High level languages are programmer friendly. They are easy to write, debug and maintain.
- It provide higher level of abstraction from machine languages.

- It is machine independent language.
- Easy to learn.
- Less error prone, easy to find and debug errors.
- High level programming results in better programming productivity.

Disadvantages of High level language

- It takes additional translation times to translate the source code to machine code.
- High level programs are comparatively slower than low level programs.
- Compared to low level programs, they are generally less memory efficient.
- It can not communicate directly to the hardware.

Unit 4.2: Data Types

Data type

- As its name indicates, data type represent a type of the data which we can process using our computer program. It can be numeric, alphanumeric, decimal, etc.
- In computer science and computer programming, a **data type** or simply **type** is an attribute of data which tells the compiler or interpreter how the programmer intends to use the data.

Let's keep Computer Programming aside for a while and take an easy example of adding two whole numbers 100 & 25, which can be done simply as follows:

100 + 25

Let's take another problem where we want to add two fractional numbers 100.50 & 25.50, which will be written as follows

100.5 + 25.5

The two examples are straight forward. Now let's take another example where we want to record student information in a notebook. Here we would like to record the information like Name, Class, Section and Age

Now, let's put one student record as per the given requirement –

Name: Ram Kumar

Class: 12th

Section: B

Age: 25

The first example dealt with whole numbers, the second example added two decimal numbers, whereas the third example is dealing with a mix of different data. Let's put it as follows –

- Student name "Ram Kumar" is a sequence of characters which is also called a string.
- Student class "12th" has been represented by a mix of whole number and a string of two characters. Such a mix is called alphanumeric.
- Student section has been represented by a single character which is 'A'.
- Student age has been represented by a whole number which is 25.

This way, we realized that in our day-to-day life, we deal with different types of data such as strings, characters, whole numbers (integers), and fractional numbers (floating point numbers).

Similarly, when we write a computer program to process different types of data, we need to specify its type clearly; otherwise the computer does not understand how different operations can be performed on that given data.

A particular kind of operations can be performed on data by seeing its data type

Data type in C

Each variable in C has an associated data type. Each data type requires different amounts of memory and has some specific operations which can be performed over it. Let us briefly describe them one by one:

Following are the examples of some very common data types used in C:

- **char:** The most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.

- **int**: As the name suggests, an int variable is used to store an integer.
- **float**: It is used to store decimal numbers (numbers with floating point value) .
- **double**: It is used to store decimal numbers (numbers with floating point value but its range of values is high in comparison to float.

Important Data type in C with range

Type	Keyword	Value range which can be represented by this data type
Character	char	-128 to 127 or 0 to 255
Number	int	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
Small Number	short	-32,768 to 32,767
Long Number	long	-2,147,483,648 to 2,147,483,647
Decimal Number	float	1.2E-38 to 3.4E+38 till 6 decimal places

These data types are called primitive data types and we can use these data types to build more complex data types, which are called user-defined data type, for example a string will be a sequence of characters.

Creating variables

Creating variables is also called **declaring variables** in C programming. Different programming languages have different ways of creating variables inside a program. For example, C programming has the following simple way of creating variables –

```
int main()
 {
  int a;
  int b;
}
```

The above program creates two variables to reserve two memory locations with names **a** and **b**.

We created these variables using **int** keyword to specify variable **data type** which means we want to store integer values in these two variables. Similarly, we can create variables to store **long**, **float**, **char** or any other data type.

Important key points about variables

- A variable name can hold a single type of value. For example, if variable **a** has been defined **int** type, then it can store only integer.
- C programming language requires a variable creation, i.e., declaration before its usage in our program. We cannot use a variable name in our program without creating it, though programming language like Python allows you to use a variable name without creating it.

- We can declare a variable name only once inside the program.

  For example, if a variable **a** has been defined to store an integer value, then we cannot define **a** again to store any other type of value.

Unit 4.3: Basic Arithmetic Operations

Arithmetic Operators in C

```c
#include <stdio.h>

int main() {
    int v = 0;
    printf("%d\n", &v);
    return 0;
}
```

Name	Operator	Example
Addition	+	a+b will give 13
Subtraction	-	a –b will give 7
Multiplication	*	a*b will give 30
Division	/	a/b will give 3
Modulus	%	m % n will give 1

Note : a=10 and b=3

Division

```c
#include <stdio.h>

int main() {
    int v = 0;
    printf("%d\n", &v);
    return 0;
}
```

- If both operands of a division expression are integers, you will get an integer answer. The fractional portion is thrown away.
- Examples : **17 / 5 = 3**
  4 / 3 = 1
  35 / 9 = 3

DIVISION

```c
#include <stdio.h>

int main() {
    int v = 0;
    printf("%d\n", &v);
    return 0;
}
```

- Division where at least one operand is a floating point number will produce a floating point answer.
- Examples : 15.0 / 4 = 3.75
  3 / 2.2 = 1.36
  3.1 / 2.0 =1.55
- Note: The integer operand is temporarily converted to a floating point, then the division is performed.

MODULUS

```c
#include <stdio.h>

int main() {
    int v = 0;
    printf("%d\n", &v);
    return 0;
}
```

- The expression **m % n gives** remainder after **m** is divided by **n**.
- Modulus is an integer operation -- both operands MUST be integers.
- Examples : 17 % 5 = 2
  6 % 3 = 0
  9 % 2 = 1
  5 % 8 = 5

Arithmetic Operators Precedence

```c
#include <stdio.h>

int main() {
    int v = 0;
    printf("%d\n", &v);
    return 0;
}
```

Operator(s)	Precedence & Associativity
( )	Evaluated first. If **nested (embedded)**, innermost evaluated first. If on same level then evaluated from left to right.
* / %	Evaluated second. If there are several, evaluated left to right
+ -	Evaluated third. If there are several, evaluated left to right.

Arithmetic Operators Precedence --- Example

```c
#include <stdio.h>

int main() {
    int v = 0;
    printf("%d\n", &v);
    return 0;
}
```

- Using this order, let's see how the expression/formula **20/(8-4)*8-2** is calculated

20/(8-4)*8-2
Perform the operations in parentheses first: 8-4=4

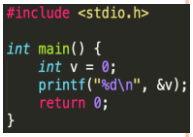
formula becomes
20/4*8-2
Because the division comes before the multiplication, divide 20/4=5

formula becomes
5*8-2
Next the multiplication takes place before the subtraction: 5*8=40

formula becomes
40-2
Finally, 40-2=38

The final answer is 38

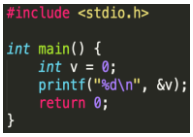
USING PARENTHESES



- Use parentheses to change the order in which an expression is evaluated.
- a + b * c Would multiply b * c first, then add a to the result.
- If you really want the sum of a and b to be multiplied by c, use parentheses to force the evaluation to be done in the order you want.

  (a + b) * c
- Also use parentheses to clarify a complex expression.

ARITHMETIC OPERATORS(CONTD.)



Example:-

```
#include <stdio.h>
int main() {
int a, b, c;
 a = 10;
 b = 20;
 c = a + b;
 printf( "Value of c = %d\n", c);
 c = a - b;
 printf( "Value of c = %d\n", c);
 c = a * b;
printf( "Value of c = %d\n", c);
c = b / a;
printf( "Value of c = %d\n", c);
c = b % a;
printf( "Value of c = %d\n", c);
}
```

When the above program is executed, it produces the following result –

```
Value of c = 30
Value of c = -10
Value of c = 200
Value of c = 2
Value of c = 0
```

RELATIONAL OPERATOR

```c
#include <stdio.h>

int main() {
    int v = 0;
    printf("%d\n", &v);
    return 0;
}
```

- Relational operators are used for comparison of the values of two operands.

  For example: checking if one operand is equal to the other operand

  if a=10 and b=10

  a==b true

If a=10 and b=20

a==b false

RELATIONAL OPERATOR

```c
#include <stdio.h>

int main() {
    int v = 0;
    printf("%d\n", &v);
    return 0;
}
```

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

RELATIONAL OPERATOR(CONTD.)

```c
#include <stdio.h>

int main() {
    int v = 0;
    printf("%d\n", &v);
    return 0;
}
```

<	Checks if the value of left operand is less than the value of right operand, if yes then 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 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 condition becomes true.	(A <= B) is true.

RELATIONAL OPERATOR

```c
#include <stdio.h>

int main() {
    int v = 0;
    printf("%d\n", &v);
    return 0;
}
```

```c
#include <stdio.h>
void main() {
 int a=10, b=20;
 /* Here we check whether a is equal to 10 or not */
 if( a == 10 ) {          /* if a is equal to 10 then this body will be executed */
printf( "a is equal to 10\n");
  }                       /* Here we check whether b is equal to 10 or not */
if( b == 10 )  {
 /* if b is equal to 10 then this body will be executed */
 printf( "b is equal to 10\n");
 }                        /* Here we check if a is less b than or not */
if( a < b )  {            /* if a is less than b then this body will be executed */
printf( "a is less than b\n");   }
 /* Here we check whether a and b are not equal */
if( a != b )
{                         /* if a is not equal to b then this body will be executed */
 printf( "a is not equal to b\n");
 }}
```

RELATIONAL OPERATOR(CONTD.)

```c
#include <stdio.h>

int main() {
   int v = 0;
   printf("%d\n", &v);
   return 0;
}
```

When the program is executed, it produces the following result –

```
a is equal to 10
a is less than b
a is not equal to b
```

Unit 4.4: Control Statements

CONTROL STATEMENTS

Control Statements:

- A program is a set of statements which are normally executed sequentilly in the order in which they are appear.

Control statements used to change the order of execution of statements based on certain conditions or repeat a group of statements until certain specified conditions are met.

This involves a kind of decision making to see whether a particular condition has occurred or not and then direct the computer to execute certain statements accordingly.

- Every high level language supports basics type of control statements:
  - Branching statements
  - Looping statements
- Branching is deciding what actions to take and looping is deciding how many times to take a certain action.

CONTROL STATEMENTS

if Statement : This is the most simple form of the branching statements.

It takes an expression in parenthesis and a statement or block of statements. if the expression is true then the statement or block of statements gets executed otherwise these statements are skipped.

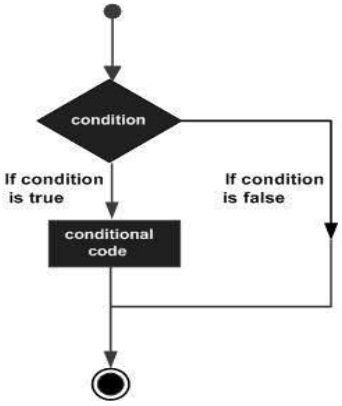
CONTROL STATEMENTS

```c
#include <stdio.h>

int main() {
    int v = 0;
    printf("%d\n", &v);
    return 0;
}
```

Syntax of If statement

if (expression)
statement;
Or

if (expression)
{
Block of statements;
}



- Almost all the programming languages provide this statement that work based on the above flow diagram

CONTROL STATEMENTS

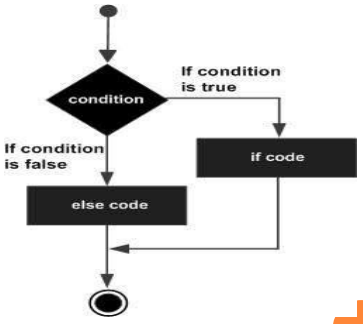
```c
#include <stdio.h>

int main() {
    int v = 0;
    printf("%d\n", &v);
    return 0;
}
```

IF … else statement:

if statement can be followed by an optional **else** block of statements, which executes when the Boolean expression is false.

syntax

if (expression)
{
true Block of statements;
}
else
{
else Block of statements;
}



CONTROL STATEMENTS

```
#include <stdio.h>

int main() {
   int v = 0;
   printf("%d\n", &v);
   return 0;
}
```

You can understand the concept of conditional statements in C programming with following program.

- For example, Find the largest of two numbers, if the numbers are a=30 and b=50

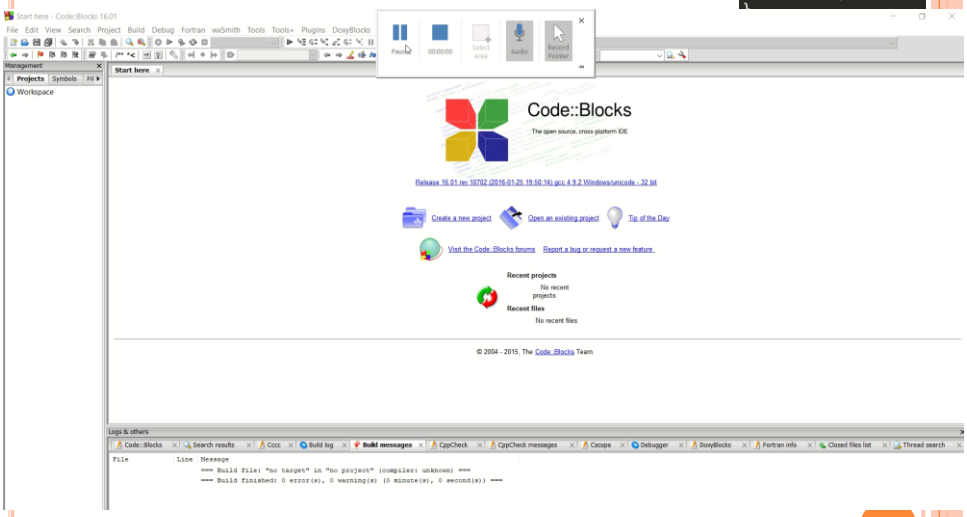
```
#include <stdio.h>
void main()
{
  int a = 30, b=50;
   if( a > b )
                printf( "Largest number is %d\n", a);
  else
     printf( "Largest number is %d\n", b);
  }
```

When the above program is executed, it produces the following result –

- Largest number is 50

CONTROL STATEMENTS

```
#include <stdio.h>

int main() {
   int v = 0;
   printf("%d\n", &v);
   return 0;
```



CONTROL STATEMENTS

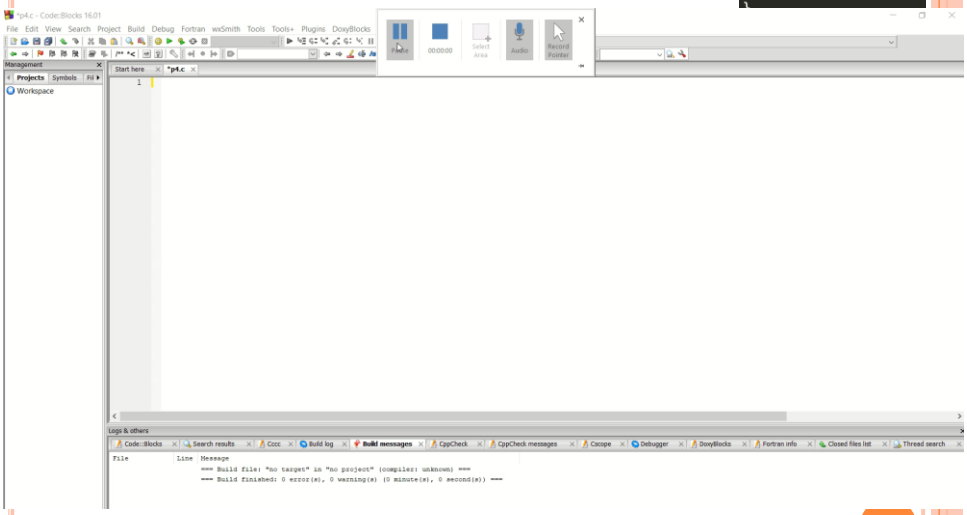
```c
#include <stdio.h>

int main() {
    int v = 0;
    printf("%d\n", &v);
    return 0;
}
```

Program to find the largest of three numbers

```c
#include <stdio.h>
void main()
{
int a, b, c, lar;
printf("Enter three numbers\n");
scanf("%d %d %d", &a, &b, &c);

if(a>b)
lar=a;
else
lar=b;

if(c>lar)
lar=c;

printf("Largest No is %d", lar);
}
```

CONTROL STATEMENTS

```c
#include <stdio.h>

int main() {
    int v = 0;
    printf("%d\n", &v);
    return 0;
}
```



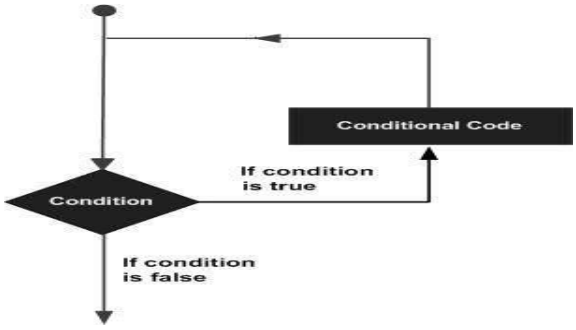
Unit 4.5: Looping Statements

Looping Statements

```c
#include <stdio.h>

int main() {
    int v = 0;
    printf("%d\n", &v);
    return 0;
}
```

Loops

- Looping statements allow us to execute a statement or group of statements multiple times.



Looping Statements


```c
#include <stdio.h>

int main() {
    int v = 0;
    printf("%d\n", &v);
    return 0;
}
```

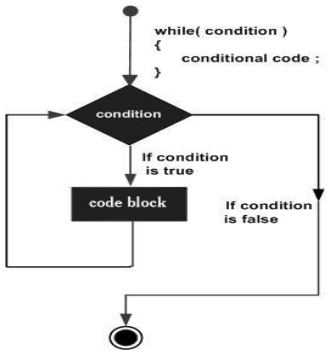

while loop:

- Basic **syntax** of while loop is as follows:

```
while( condition )
        single statement;
```

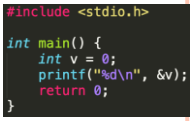
OR

```
while( condition )
{
block of statements;
}
```



The above code can be represented in the form of a flow diagram as shown above

Looping Statements



Trace while loop:

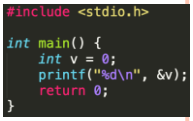
```
int count =0;
 while(count<2)
{
 printf("This is my first program\n");
count=count+1;
}
```

Output

This is my first program

This is my first program

Looping Statements



for loop

for loop is similar to while. Basic syntax of **for loop** is as follows:

```
for( expression1; expression2; expression3)
{
Block of statements;
}
```

In the above syntax:

- expression1 - Initializes variables.
- expression2 - Conditional expression, as long as this condition is true, loop will keep executing.
- expression3 – expression3 is the modifier which will increase or decrease the value of the variable.

Looping Statements

```c
#include <stdio.h>

int main() {
    int v = 0;
    printf("%d\n", &v);
    return 0;
}
```

Trace for Loop

```
int count;
 for(count=0; count<3; count++)
{
 printf("This is my first program\n")
}
```

Output

This is my first program

This is my first program

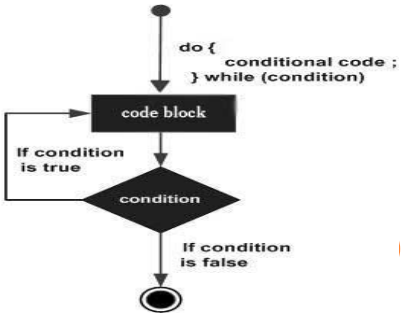
Looping Statements


```c
#include <stdio.h>

int main() {
    int v = 0;
    printf("%d\n", &v);
    return 0;
}
```


- **do...while loop**
- **do ... while** is just like a while loop except that the test condition is checked at the end of the loop rather than the start. This has the effect that the body of the loop are always executed at least once.
- **Basic syntax of do...while loop is as follows:**

```
do
{
single statement
        or
Block of statements
}while(condition);
```

This code can be represented in the form of a flow diagram as shown below



LOOPING STATEMENTS

```c
#include <stdio.h>

int main() {
    int v = 0;
    printf("%d\n", &v);
    return 0;
}
```

Trace do-while loop

```
int count =0;
 do
{
 printf("This is my first program\n")
count=count+1;
}while(count<3);
```

Output

This is my first program

This is my first program

LOOPING STATEMENTS

```c
#include <stdio.h>

int main() {
    int v = 0;
    printf("%d\n", &v);
    return 0;
}
```

You can understand the concept of the various loops by execution of the following **C** programming

Using for loop	Using while loop	Using do…while loop

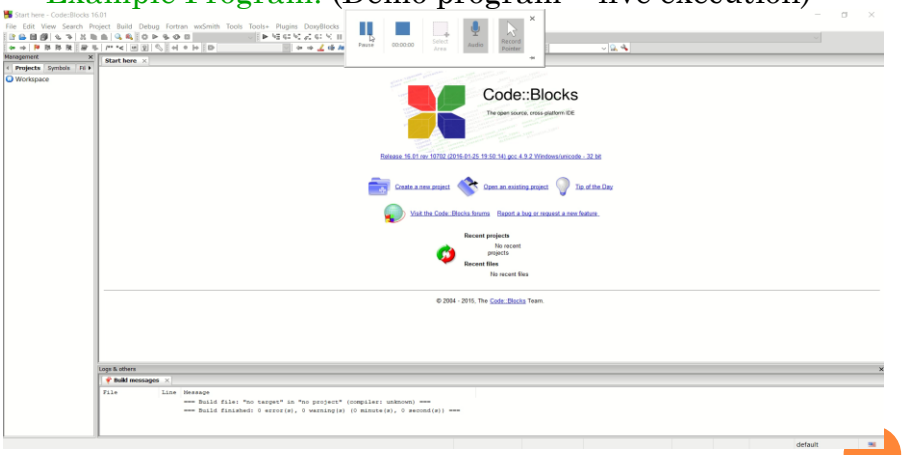
#include <stdio.h>
 	
int main()
{
int i;
for (i=1; i<=5; i++)
{
printf("Hello\n");
}
}
	#include <stdio.h>
	
	int main()
	{
	int i = 1;
	while (i <= 5)
	{
	printf("Hello\n");
	i = i + 1;
	}
	}
		#include <stdio.h>
		int main()
		{
		int i = 1;
		do
		{
		printf("Hello \n");
		i = i + 1;
		}while (i <=5);
		}

Output Hello Hello Hello Hello Hello	Output Hello Hello Hello Hello Hello	Output Hello Hello Hello Hello Hello

LOOPING STATEMENTS

```c
#include <stdio.h>

int main() {
    int v = 0;
    printf("%d\n", &v);
    return 0;
}
```

Example Program: (Demo program – live execution)



LOOPING STATEMENTS

```c
#include <stdio.h>

int main() {
    int v = 0;
    printf("%d\n", &v);
    return 0;
}
```

break statement:

It is a special statement that can *affect* the execution of loop statements (such as a *while, for & do-while-statements*).

When the **break** statement is executed inside a loop, the loop is immediately terminated and

The execution of the program will continue with the next statement *following* the loop.

The syntax for a **break** statement in C is as follows

```
break;
```

```
while ( loop-continuation-condition )
{
   statement1
   statement2
   ....
   break; ~
   ....
}                    Execution proceeds to here
statement  <--------  statement following the while loop
```

LOOPING STATEMENTS

```c
#include <stdio.h>

int main() {
    int v = 0;
    printf("%d\n", &v);
    return 0;
}
```

Break and continue:

You can understand the concept of the break and continue statements by execution of the following programs in C programming.

Break statement	Continue statement

#include <stdio.h>
void main()
{
int i = 0;
while(i<5)
{
printf("Hello %d\n", i);
i = i + 1;
if(i == 2) break;
}
}
	#include <stdio.h>
	void main()
	{
	int i = 0;
	while(i<5)
	{ i=i+1;
	if(i == 2) continue;
	printf("Hello %d\n", i);
	}
	}

When the above program is executed, it produces the following result – Hello 0 Hello 1	When the above program is executed, it produces the following result – Hello 1 Hello 3 Hello 4 Hello 5

LOOPING STATEMENTS

```c
#include <stdio.h>

int main() {
    int v = 0;
    printf("%d\n", &v);
    return 0;
}
```

Example Program: (Demo program – live execution)