Module –I  
C Language Fundamentals.  
Character set, Identifiers, keyword, data types, Constants and variables, statements,   
expression, operators, precedence of operators, Input-output, Assignments, control structures  
decision making and branching.  
Module -II  
Arrays, Functions and Strings: Declaration, manipulation and String – handling   
functions, monolithic vs. Modular programs, user defined vs. standard functions, formal vs. actual   
arguments, function – category, function prototypes, parameter passing, recursion, and storage classes:   
auto, extern, global, static.  
Module –III  
Pointers, Structures, Unions, File handling:  
Pointer variable and its importance, pointer arithmetic, passing parameters, Declaration of structures,   
pointer to pointer, pointer to structure, pointer to function, union, dynamic memory allocation, file   
managements.

CONTENTS   
Module: 1   
Session 1: Introduction to C  
Session 2: Structure of C, compilation, execution   
Session 3:character set, identifiers, keywords   
Session 4: constants, variables  
Session 5: expression, operators   
Session 6: operators continue...  
Session 7: loops: do while, while  
Session 8: for loop, break, continue statement  
Session 9: control Statements  
Session 10: nesting of if else..., if else ladder  
Session 11: arrays  
Session 12: 2-diamensional array

Module: 2   
Session 13: String library functions  
Session 14: functions, categories   
Session 15: functions categories cont..   
Session 16: Actual arguments and Formal arguments, call by value call by   
reference  
Session 17:local, global, static variable   
Session 18: monolithic vs modular programming, Storage classes  
Session 19:storage class cont.., pointer  
Session 20: pointer comparison, increment decrement   
Session 21: precedence level of pointer, pointer comparison  
Session 22: pointer to pointer, pointer to structure   
Session 23: pointer initialization, accessing elements

Module: 3  
Session 24: size of Structure in, array vs structure, array within structure   
Session 25: passing structure to function, Nested Structure  
Session 26: Union  
Session 27: nesting of unions, dynamic memory allocation  
Session 28: dynamic memory allocation conti...   
Session 29: dynamic array, file   
Session 30: file operation  
Session 31: file operation on string

In programming, a library is a collection of pre-compiled pieces of code that can be reused in a program. Libraries simplify life for programmers, in that they provide reusable functions, routines, classes, data structures and so on  
which they can be reused in the programs.

**Static Libraries :** A Static library or statically-linked library is a set of routines, external functions and variables which are resolved in a caller at compile-time and copied into a target application by a compiler, linker, or binder, producing an object file and a stand-alone executable. This executable and the process of compiling it are both known as a static build of the program. Historically, libraries could only be static.  
They are usually faster than the shared libraries because a set of commonly used object files is put into a single library executable file. One can build multiple executables without the need to recompile the file. Because it is a single file to be built, use of link commands are simpler than shared library link commands, because you specify the name of the static library.

**Shared Libraries :**  
Shared libraries are .so (or in Windows .dll, or in OS X .dylib) files.  
These are linked dynamically simply including the address of the library (whereas static linking is a waste of space). Dynamic linking links the libraries at the run-time. Thus, all the functions are in a special place in memory space, and every program can access them, without having multiple copies of them.

| **properties** | **Static library** | **Shared library** |
| --- | --- | --- |
| Linking time | It happens as the last step of the compilation process. After the program is placed in the memory | Shared libraries are added during linking process when executable file and libraries are added to the memory. |
| Means | Performed by linkers | Performed by operating System |
| Size | Static libraries are much bigger in size, because external programs are built in the executable file. | Dynamic libraries are much smaller, because there is only one copy of dynamic library that is kept in memory. |
| External file changes | Executable file will have to be recompiled if any changes were applied to external files. | In shared libraries, no need to recompile the executable. |
| Time | Takes longer to execute, because loading into the memory happens every time while executing. | It is faster because shared library code is already in the memory. |
| Compatibility | Never has compatibility issue, since all code is in one executable module. | Programs are dependent on having a compatible library. Dependent program will not work if library gets removed from the system. |

**Tokens in C**

**6 Types of Tokens in C**

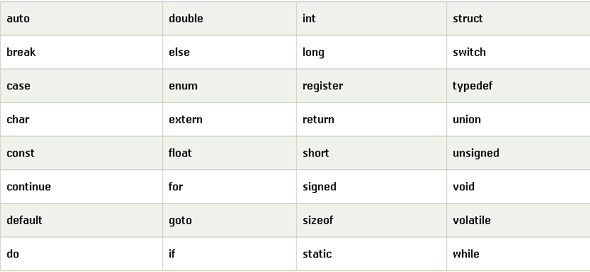
C Supports 6 Types of Tokens

* Keywords
* Identifiers
* Strings
* Operators
* Constants
* Special Symbols

**1. Keywords**

Keywords in C language are predefined or reserved keywords used to expose the behaviour of the data. There are 32 keywords in C. Each keyword has its functionality to do.

**Syntax:**



**2. Identifier**

Identifier in C language is used for naming functions, variables, structures, unions, arrays, etc. The identifier is user-defined words. These identifiers can be composed of uppercase, lowercase letters, digits, underscore. Identifiers never used for keywords. Rules to construct identifiers is below

* The first character should be either alphabet or underscore and then followed by any character, digit.
* Identifiers are case sensitive as there is **A** and **a** treated as different.
* Commas and blank space are not allowed
* Keywords can’t be used for identifiers.
* The length of the identifiers should not be more than 31 characters.
* Naming convention should understandable to the user.

**Syntax:**

dataType \_abc1= Valid  
dataType 123abcZ=Invalid  
dataType int=Invalid  
dataType abc, ap=Invalid

**3. Strings**

Strings in C is an array of characters having null character ‘\0’ at the end of the string. Strings in C are enclosed in double-quotes (“”) and Characters are enclosed in single quotes (”).

**Syntax:**

char a[10]={'1','2','3'};  
char a[]="Amardeep";  
char a[10]="Paramesh";

**4. Operators**

This is used to perform special operations on data.

**Unary Operator:** Applied with a single operand.

**Binary Operator:** Applied between 2 operands.

* Arithmetic Operators
* Relational Operators
* Shift Operators
* Logical Operators
* Bitwise Operators
* Conditional Operators
* Assignment Operator
* Misc Operator

**Ternary operator:** Applied to 3 operands.

**Syntax:** <condition>?<value if condition is true>:<value if condition is false>

Ex: Biggest=(5>2)? 5 : 2;

Here, 5 is stored in variable Biggest as the condition is true.

**5. Constants**

A constant in C language is used to make the value fixed, we can’t change constant value.

Popular Course in this category

There are 2 ways of declaring a constant:

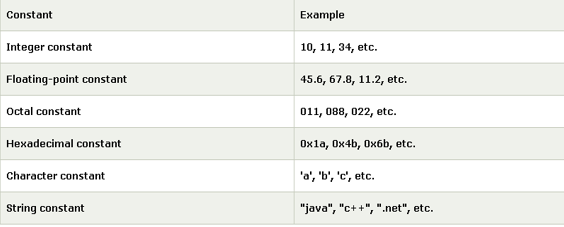
**1.** Using const keyword

const variableName;

**2.** By Using #define pre-processor

#define NAME value;

**Types of Constants**



**6. Special Symbols**

* **Square brackets [ ]:** Used for single and multi-dimensional arrays.
* **Simple brackets ( ):** Used for function declaration.
* **Curly braces { }:** Used for opening and closing the code.
* **The comma (,):** Used to separate variables.
* **Hash/pre-processor (#):** Used for the header file.
* **Asterisk (\*):** Used for Pointers.
* **Tilde (~):** Used for destructing the memory.
* **Period (.):** Used for accessing union members.

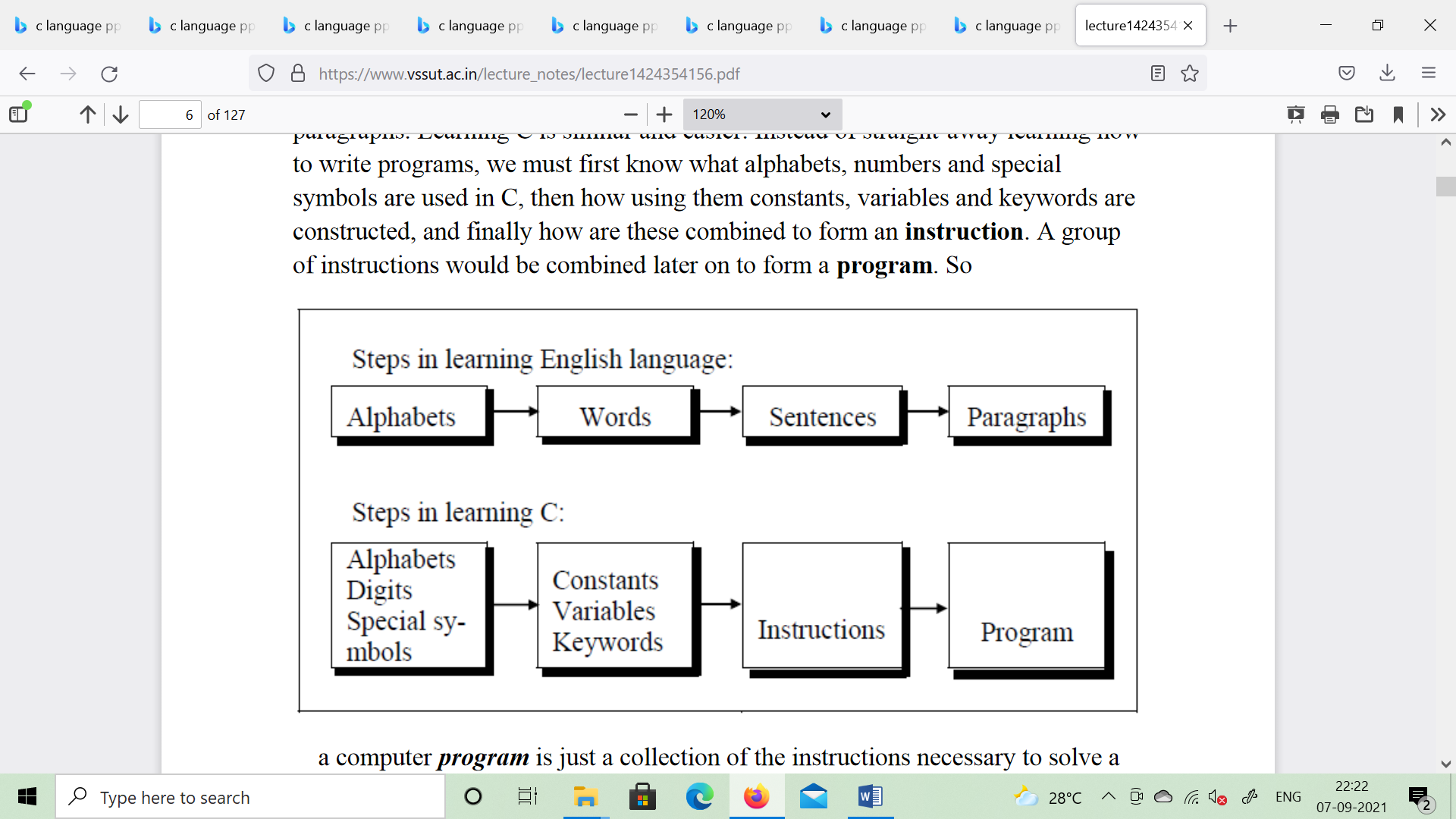
**List all the Escape Sequence Characters in C**

|  |  |  |
| --- | --- | --- |
| **Escape Sequence** | **Meaning** | **Explanation** |
| \a | Alarm or Beep | A beep sound is generated |
| \b | Backspace | Backspace |
| \f | Form Feed | Form Feed page break(Return) |
| \n | New Line | Shift the cursor control to the new line |
| \r | Carriage Return | Shift the cursor to the beginning of the current line |
| \t | Tab (Horizontal) | Shift the cursor to a couple of spaces(Eight blank spaces) to the right in the same line |
| \v | Vertical Tab | Vertical Tab |
| \\ | Backslash | Print the backslash character |
| \’ | Single Quote | Display the single-quotation mark. |
| \” | Double Quote | Print the double-quotation mark |
| \? | Question Mark | Display the question mark |
| \nnn | octal number | Represent an octal number |
| \xhh | hexadecimal number | Represent a hexadecimal number |
| \0 | Null | Termination of the string |

Session Note: 1  
  
**Introduction to C**C is a programming language developed at AT & T’s Bell Laboratories of USA   
in 1972. It was designed and written by a man named Dennis Ritchie. In the late   
seventies C began to replace the more familiar languages of that time like PL/I,   
ALGOL, etc

ANSI C standard emerged in the early 1980s, this book was split into two   
titles: The original was still called Programming in C, and the title that covered   
ANSI C was called Programming in ANSI C. This was done because it took   
several years for the compiler vendors to release their ANSI C compilers and for   
them to become ubiquitous. It was initially designed for programming UNIX   
operating system. Now the software tool as well as the C compiler is written in C.   
Major parts of popular operating systems like Windows, UNIX, Linux is still   
written in C. This is because even today when it comes to performance (speed of   
execution) nothing beats C. Moreover, if one is to extend the operating system to   
work with new devices one needs to write device driver programs. These   
programs are exclusively written in C. C seems so popular is because it is reliable,   
simple and easy to use. often heard today is – “C has been already superceded   
by languages like C++, C# and Java.

There is a close analogy between learning English language and learning C   
language. The classical method of learning English is to first learn the alphabets   
used in the language, then learn to combine these alphabets to form words, which   
in turn are combined to form sentences and sentences are combined to form   
paragraphs. Learning C is similar and easier. Instead of straight-away learning how   
to write programs, we must first know what alphabets, numbers and special   
symbols are used in C, then how using them constants, variables and keywords are   
constructed, and finally how are these combined to form an instruction. A group   
of instructions would be combined later on to form a program.



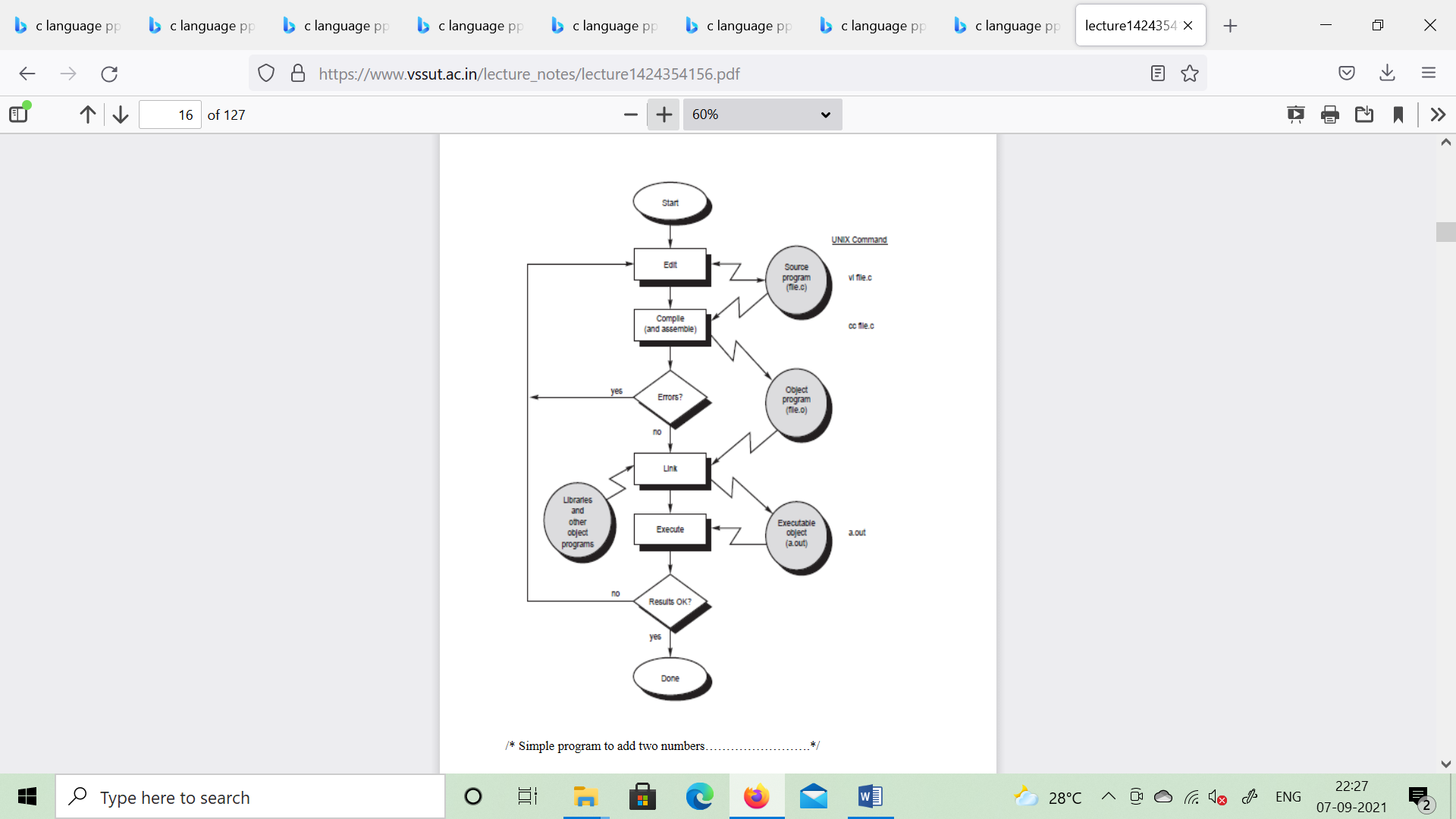
So a computer program is just a collection of the instructions necessary to solve a   
specific problem. The basic operations of a computer system form what is known   
as the computer’s instruction set. And the approach or method that is used to solve   
the problem is known as an algorithm.  
So for as programming language concern these are of two types.  
1) Low level language  
2) High level language

**Low level language**:  
Low level languages are machine level and assembly level language. In   
machine level language computer only understand digital numbers i.e. in the form   
of 0 and 1. So, instruction given to the computer is in the form binary digit, which   
is difficult to implement instruction in binary code. This type of program is not   
portable, difficult to maintain and also error prone. The assembly language is on   
other hand modified version of machine level language. Where instructions are   
given in English like word as ADD, SUM, MOV etc. It is easy to write and   
understand but not understand by the machine. So the translator used here is   
assembler to translate into machine level. Although language is bit easier,   
programmer has to know low level details related to low level language. In the   
assembly level language the data are stored in the computer register, which varies   
for different computer. Hence it is not portable.   
**High level language**:  
These languages are machine independent, means it is portable. The language in   
this category is Pascal, Cobol, Fortran etc. High level languages are understood by   
the machine. So it need to translate by the translator into machine level. A   
translator is software which is used to translate high level language as well as low   
level language in to machine level language.  
Three types of translator are there:  
Compiler  
Interpreter   
Assembler  
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. Both compiler and interpreter perform the same task but there working is   
different. 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. If it check error, statement by   
statement and hence of take more time.

**Integrated Development Environments (IDE)**  
The process of editing, compiling, running, and debugging programs is often   
managed by a single integrated application known as an Integrated Development   
Environment, or IDE for short. An IDE is a windows-based program that allows us   
to easily manage large software programs, edit files in windows, and compile, link,   
run, and debug programs.  
On Mac OS X, CodeWarrior and Xcode are two IDEs that are used by many   
programmers. Under Windows, Microsoft Visual Studio is a good example of a   
popular IDE. Kylix is a popular IDE for developing applications under Linux.   
Most IDEs also support program development in several different programming   
languages in addition to C, such as C# and C++.

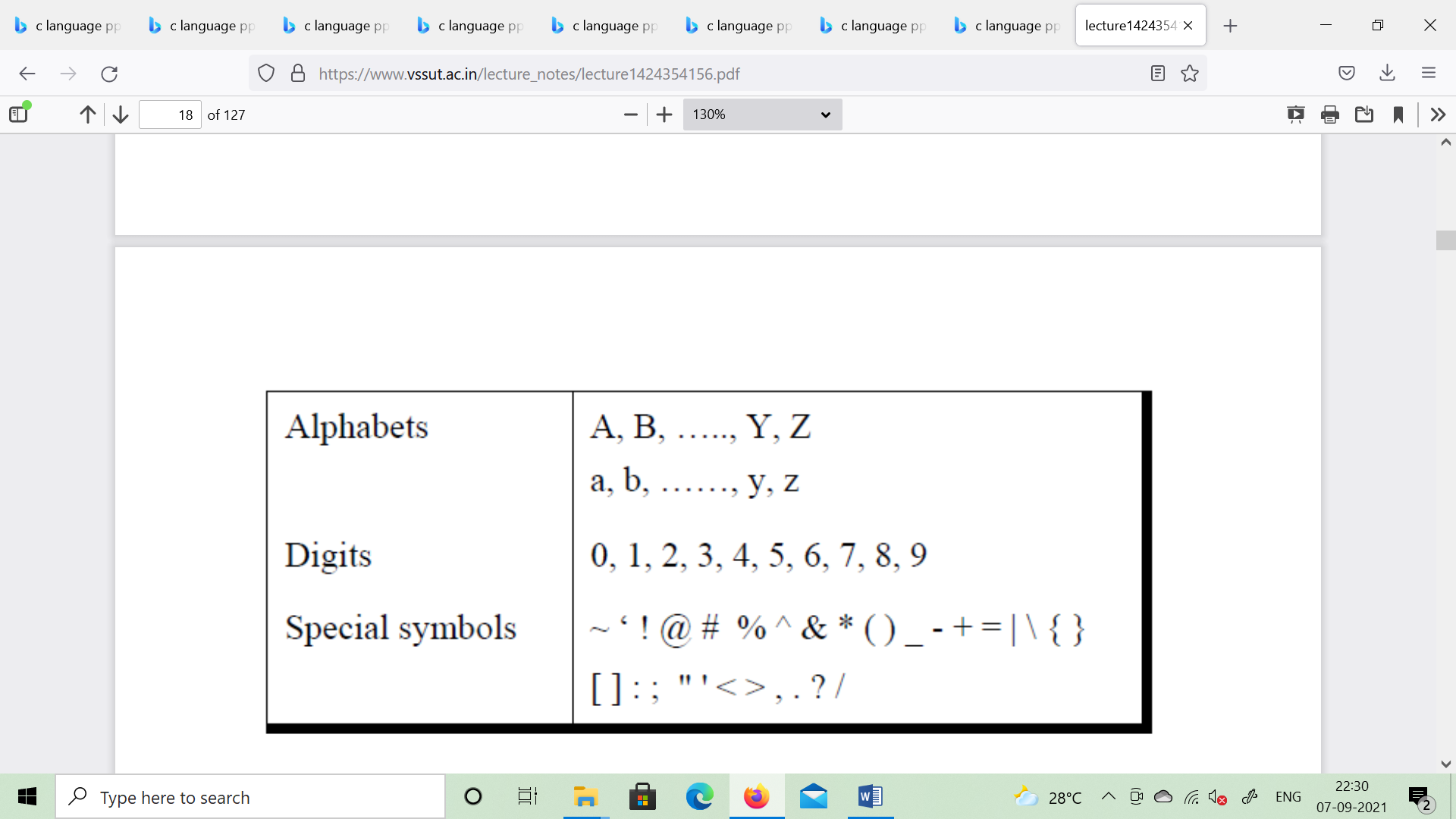
Session Note: 2  
Structure of C Language program  
1 ) Comment line  
2) Preprocessor directive  
3 ) Global variable declaration  
4) main function( )  
{  
Local variables;  
Statements;  
}  
User defined function  
}  
}  
**Comment line**  
It indicates the purpose of the program. It is represented as  
/\*...................................\*/   
Comment line is used for increasing the readability of the program. It is useful in   
explaining the program and generally used for documentation. It is enclosed within   
the decimeters. Comment line can be single or multiple line but should not be   
nested. It can be anywhere in the program except inside string constant & character   
constant.  
**Preprocessor Directive**:  
#include<stdio.h> tells the compiler to include information about the standard   
input/output library. It is also used in symbolic constant such as #define PI   
3.14(value). The stdio.h (standard input output header file) contains definition   
&declaration of system defined function such as printf( ), scanf( ), pow( ) etc.   
Generally printf() function used to display and scanf() function used to read value  
Global Declaration:  
This is the section where variable are declared globally so that it can be access by   
all the functions used in the program. And it is generally declared outside the   
function :  
main()  
It is the user defined function and every function has one main() function from   
where actually program is started and it is encloses within the pair of curly braces.  
The main( ) function can be anywhere in the program but in general practice it is   
placed in the first position.  
Syntax :  
main()  
{  
........  
........  
........  
}  
The main( ) function return value when it declared by data type as   
int main( )   
{  
return 0

}  
The main function does not return any value when void (means null/empty) as  
void main(void ) or void main()  
{  
printf (“C language”);  
}  
Output: C language  
The program execution start with opening braces and end with closing brace.  
And in between the two braces declaration part as well as executable part is   
mentioned. And at the end of each line, the semi-colon is given which indicates   
statement termination.   
/\*First c program with return statement\*/  
#include <stdio.h>  
int main (void)  
{  
printf ("welcome to c Programming language.\n");  
return 0;  
}  
Output: welcome to c programming language.  
Steps for Compiling and executing the Programs  
A compiler is a software program that analyzes a program developed in a particular   
computer language and then translates it into a form that is suitable for execution   
on a particular computer system. Figure below shows the steps that are involved in   
entering, compiling, and executing a  
computer program developed in the C programming language and the typical Unix   
commands that would be entered from the command line.   
Step 1: The program that is to be compiled is first typed into a file on the   
computer system. There are various conventions that are used for naming files,   
typically be any name provided the last two characters are “.c” or file with   
extension .c. So, the file name prog1.c might be a valid filename for a C program.   
A text editor is usually used to enter the C program into a file. For example, vi is a   
popular text editor used on Unix systems. The program that is entered into the file   
is known as the source program because it represents the original form of the   
program expressed in the C language.   
Step 2: After the source program has been entered into a file, then proceed to have   
it compiled. The compilation process is initiated by typing a special command on   
the system. When this command is entered, the name of the file that contains the   
source program must also be specified. For example, under Unix, the command to   
initiate program compilation is called cc. If we are using the popular GNU C   
compiler, the command we use is gcc.  
Typing the line  
gcc prog1.c or cc prog1.c  
In the first step of the compilation process, the compiler examines each program  
statement contained in the source program and checks it to ensure that it conforms   
to the syntax and semantics of the language. If any mistakes are discovered by the   
compiler during this phase, they are reported to the user and the compilation   
process ends right there. The errors then have to be corrected in the source program   
(with the use of an editor), and the compilation process must be restarted. Typical   
errors reported during this phase of compilation might be due to an expression that   
has unbalanced parentheses (syntactic error), or due to the use of a variable that is   
not “defined” (semantic error).  
Step 3: When all the syntactic and semantic errors have been removed from the   
program, the compiler then proceeds to take each statement of the program and   
translate it into a “lower” form that is equivalent to assembly language program   
needed to perform the identical task.  
Step 4: After the program has been translated the next step in the compilation   
process is to translate the assembly language statements into actual machine   
instructions. The assembler takes each assembly language statement and converts it   
into a binary format known as object code, which is then written into another file   
on the system. This file has the same name as the source file under Unix, with the   
last letter an “o” (for object) instead of a “c”.   
Step 5: After the program has been translated into object code, it is ready to be   
linked. This process is once again performed automatically whenever the cc or gcc   
command is issued under Unix. The purpose of the linking phase is to get the   
program into a final form for execution on the computer.  
If the program uses other programs that were previously   
processed by the compiler, then during this phase the programs are linked together.   
Programs that are used from the system’s program library are also searched and   
linked together with the object program during this phase.  
The process of compiling and linking a program is often called building.  
The final linked file, which is in an executable object code format, is stored in   
another file on the system, ready to be run or executed. Under Unix, this file is   
called a.out by default. Under Windows, the executable file usually has the same   
name as the source file, with the c extension replaced by an exe extension.  
Step 6: To subsequently execute the program, the command a.out has the effect   
of loading the program called a.out into the computer’s memory and initiating its   
execution.  
When the program is executed, each of the statements of the program is   
sequentially executed in turn. If the program requests any data from the user,   
known as input, the program temporarily suspends its execution so that the input   
can be entered. Or, the program might simply wait for an event, such as a mouse   
being clicked, to occur. Results that are displayed by the program, known as   
output, appear in a window, sometimes called the console. If the program does not   
produce the desired results, it is necessary to go back and reanalyze the program’s   
logic. This is known as the debugging phase, during which an attempt is made to   
remove all the known problems or bugs from the program. To do this, it will most   
likely be necessary to make changes to original source program.



#include <stdio.h>  
int main (void)  
{  
int v1, v2, sum; //v1,v2,sum are variables and int is data type declared  
v1 = 150;  
v2 = 25;  
sum = v1 + v2;  
printf ("The sum of %i and %i is= %i\n", v1, v2, sum);  
return 0;  
}  
Output:  
The sum of 150 and 25 is=175

**Character set**  
A character denotes any alphabet, digit or special symbol used to represent   
information. Valid alphabets, numbers and special symbols allowed in C are

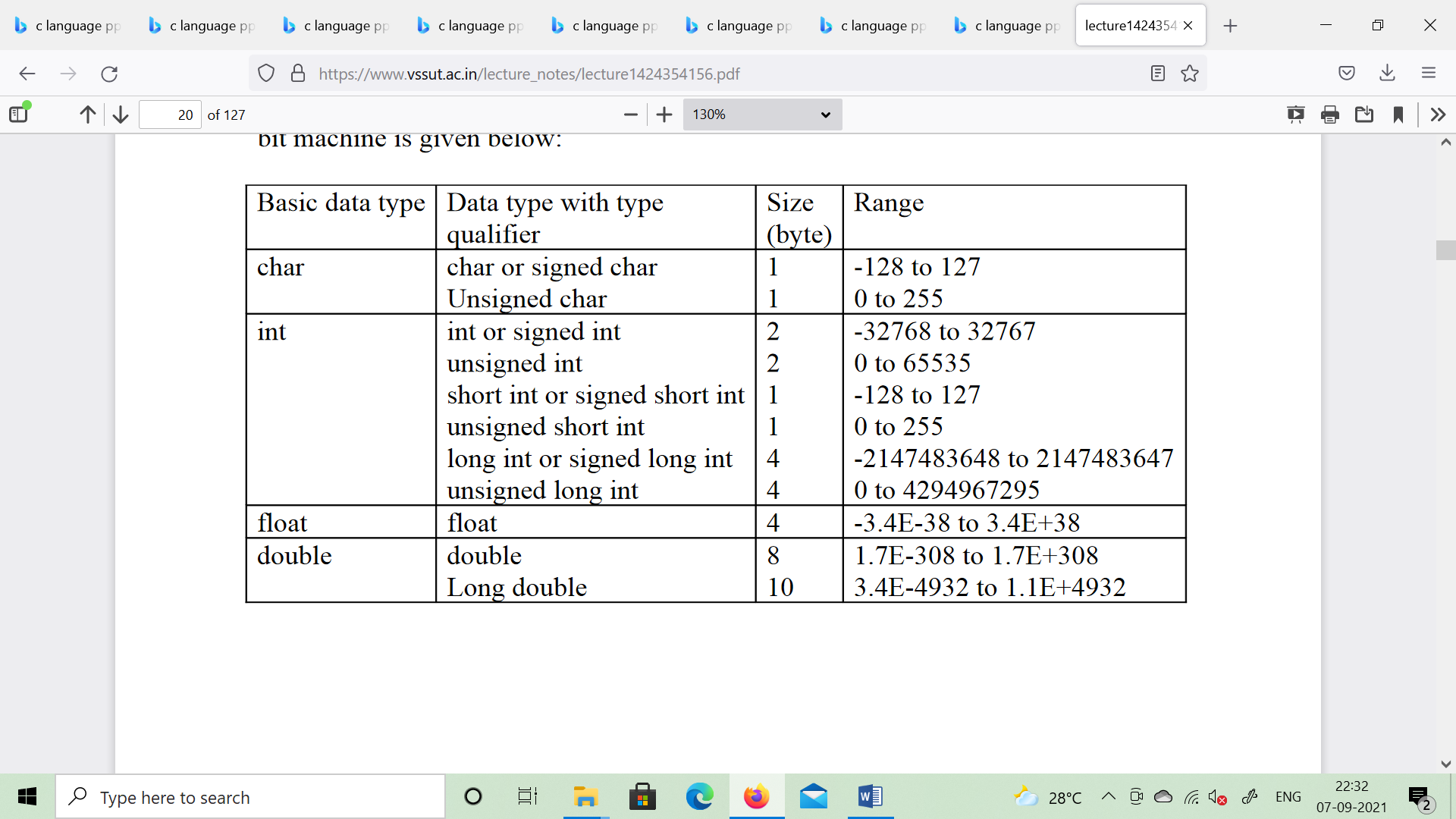


The alphabets, numbers and special symbols when properly combined form   
constants, variables and keywords.   
Identifiers  
Identifiers are user defined word used to name of entities like variables, arrays,   
functions, structures etc. Rules for naming identifiers are:  
1) name should only consists of alphabets (both upper and lower case), digits   
and underscore (\_) sign.  
2) first characters should be alphabet or underscore  
3) name should not be a keyword  
4) since C is a case sensitive, the upper case and lower case considered   
differently, for example code, Code, CODE etc. are different identifiers.  
5) identifiers are generally given in some meaningful name such as value,   
net\_salary, age, data etc. An identifier name may be long, some implementation   
recognizes only first eight characters, most recognize 31 characters. ANSI   
standard compiler recognize 31 characters. Some invalid identifiers are 5cb, int,   
res#, avg no etc.

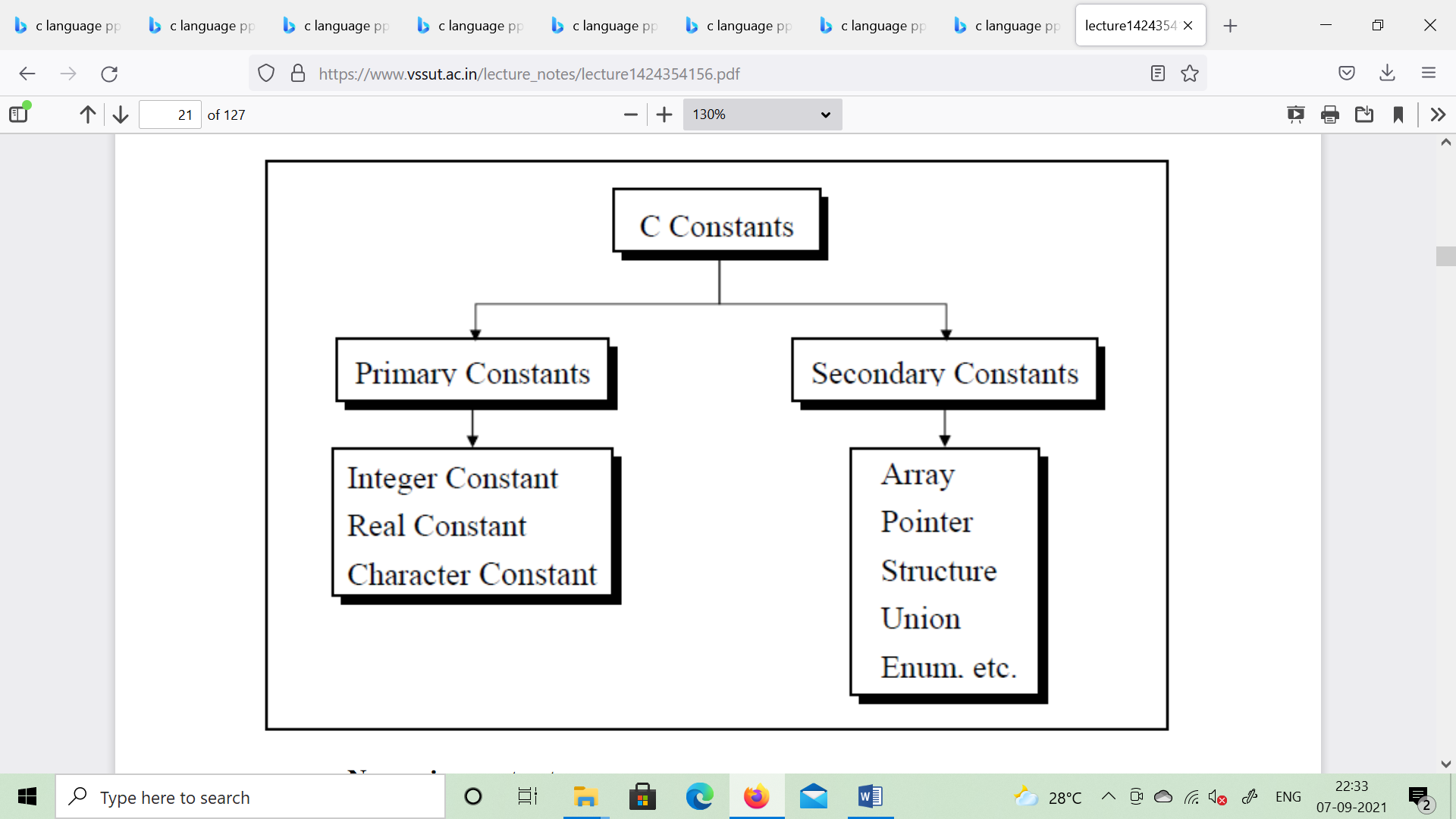
**Keyword**

There are certain words reserved for doing specific task, these words   
are known as reserved word or keywords. These words are predefined and always   
written in lower case or small letter. These keywords cann’t be used as a variable   
name as it assigned with fixed meaning. Some examples are int, short, signed,   
unsigned, default, volatile, float, long, double, break, continue, typedef, static,   
do, for, union, return, while, do, extern, register, enum, case, goto, struct,   
char, auto, const etc.   
data types  
Data types refer to an extensive system used for declaring variables or functions of   
different types before its use. The type of a variable determines how much space it   
occupies in storage and how the bit pattern stored is interpreted. The value of a   
variable can be changed any time.  
C has the following 4 types of data types  
basic built-in data types: int, float, double, char  
Enumeration data type: enum  
Derived data type: pointer, array, structure, union  
Void data type: void  
A variable declared to be of type int can be used to contain integral values   
only—that is, values that do not contain decimal places. A variable declared to be   
of type float can be used for storing floating- point numbers (values containing   
decimal places). The double type is the same as type float, only with roughly twice   
the precision. The char data type can be used to store a single character, such as the   
letter a, the digit character 6, or a semicolon similarly A variable declared char can   
only store character type value.   
There are two types of type qualifier in c  
Size qualifier: short, long  
Sign qualifier: signed, unsigned

When the qualifier unsigned is used the number is always positive, and when   
signed is used number may be positive or negative. If the sign qualifier is not   
mentioned, then by default sign qualifier is assumed. The range of values for   
signed data types is less than that of unsigned data type. Because in signed type,   
the left most bit is used to represent sign, while in unsigned type this bit is also   
used to represent the value. The size and range of the different data types on a 16   
bit machine is given below:



**Constants**   
Constant is a any value that cannot be changed during program execution. In C,   
any number, single character, or character string is known as a constant. A constant   
is an entity that doesn’t change whereas a variable is an entity that may change.   
For example, the number 50 represents a constant integer value. The character   
string "Programming in C is fun.\n" is an example of a constant character string. C   
constants can be divided into two major categories:   
Primary Constants   
**Secondary Constants**These constants are further categorized as



**Numeric constant**

**Character constant**

**String constant**

**Numeric constant:** Numeric constant consists of digits. It required minimum size   
of 2 bytes and max 4 bytes. It may be positive or negative but by default sign is   
always positive. No comma or space is allowed within the numeric constant and it   
must have at least 1 digit. The allowable range for integer constants is -32768 to   
32767. Truly speaking the range of an Integer constant depends upon the compiler.   
For a 16-bit compiler like Turbo C or Turbo C++ the range is –32768 to 32767.   
For a 32-bit compiler the range would be even greater. Mean by a 16-bit or a 32-  
bit compiler, what range of an Integer constant has to do with the type of compiler.   
It is categorized a integer constant and real constant. An integer constants are   
whole number which have no decimal point.

**Types of integer constants are:  
Decimal constant: 0-------9(base 10)   
Octal constant: 0-------7(base 8)  
Hexa decimal constant: 0----9, A------F(base 16)**In decimal constant first digit should not be zero unlike octal constant first digit   
must be zero(as 076, 0127) and in hexadecimal constant first two digit should be   
0x/ 0X (such as 0x24, 0x87A). By default type of integer constant is integer but if   
the value of integer constant is exceeds range then value represented by integer   
type is taken to be unsigned integer or long integer. It can also be explicitly   
mention integer and unsigned integer type by suffix l/L and u/U.   
**Real constant is also called floating point constant. To construct real constant we   
must follow the rule of ,   
-real constant must have at least one digit.   
-It must have a decimal point.   
-It could be either positive or negative.   
-Default sign is positive.   
-No commas or blanks are allowed within a real constant. Ex.: +325.34   
426.0   
-32.76   
To express small/large real constant exponent(scientific) form is used where   
number is written in mantissa and exponent form separated by e/E. Exponent can   
be positive or negative integer but mantissa can be real/integer type, for example   
3.6\*105=3.6e+5. By default type of floating point constant is double, it can also be   
explicitly defined it by suffix of f/F.**

**Character constant**Character constant represented as a single character enclosed within a single   
quote. These can be single digit, single special symbol or white spaces such as   
‘9’,’c’,’$’, ‘ ’ etc. Every character constant has a unique integer like value in   
machine’s character code as if machine using ASCII (American standard code for   
information interchange). Some numeric value associated with each upper and   
lower case alphabets and decimal integers are as:  
**A------------ Z ASCII value (65-90)  
a-------------z ASCII value (97-122)  
0-------------9 ASCII value (48-59)  
; ASCII value (59)**

**String constant**Set of characters are called string and when sequence of characters are   
enclosed within a double quote (it may be combination of all kind of symbols) is a   
string constant. String constant has zero, one or more than one character and at the   
end of the string null character(\0) is automatically placed by compiler. Some   
examples are “sarathina” , “908”, “3”,” ”, “A” etc. In C although same characters   
are enclosed within single and double quotes it represents different meaning such   
as “A” and ‘A’ are different because first one is string attached with null character   
at the end but second one is character constant with its corresponding ASCII value   
is 65.

**Symbolic constant**Symbolic constant is a name that substitute for a sequence of characters and,   
characters may be numeric, character or string constant. These constant are   
generally defined at the beginning of the program as   
#define name value , here name generally written in   
upper case for example

#define MAX 10  
#define CH ‘b’  
#define NAME “sony”

Function free() is used to release space allocated dynamically, the memory   
released by free() is made available to heap again. It can be used for further   
purpose.  
Syntax for free declaration .  
void(\*ptr)  
Or  
free(p)  
When program is terminated, memory released automatically by the operating   
system. Even we don’t free the memory, it doesn’t give error, thus lead to memory   
leak.  
We can’t free the memory, those didn’t allocated.  
Lecture Note: 29  
Dynamic array  
Array is the example where memory is organized in contiguous way, in the   
dynamic memory allocation function used such as malloc(), calloc(), realloc()   
always made up of contiguous way and as usual we can access the element in two   
ways as:  
Subscript notation  
Pointer notation  
Example:  
#include<stdio.h>  
#include<alloc.h>  
void main()  
{  
printf(“enter the no.of values”);  
scanf(“%d”,&n);  
p=(int\*)malloc(n\*size of int);  
If(p==null)  
printf(“not available memory”);  
exit();  
}  
for(i=0;i<n;i++)  
{  
printf(“enter an integer”);  
scanf(“%d”,&p[i]);  
for(i=0;i<n;i++)  
{  
printf(“%d”,p[i]);  
}  
}  
File handling  
File: the file is a permanent storage medium in which we can store the data   
permanently.  
Types of file can be handled  
we can handle three type of file as   
(1) sequential file   
(2) random access file   
(3) binary file   
File Operation  
opening a file:   
Before performing any type of operation, a file must be opened and for this   
fopen() function is used.  
syntax:  
file-pointer=fopen(“FILE NAME ”,”Mode of open”);  
example:  
FILE \*fp=fopen(“ar.c”,”r”);  
If fopen() unable to open a file than it will return NULL to the file pointer.  
File-pointer: The file pointer is a pointer variable which can be store the address   
of a special file that means it is based upon the file pointer a file gets opened.  
Declaration of a file pointer:-  
FILE\* var;  
Modes of open  
The file can be open in three different ways as   
Read mode ’ r’/rt  
Write mode ’w’/wt  
Appened Mode ’a’/at  
Reading a character from a file  
getc() is used to read a character into a file   
Syntax:  
character\_variable=getc(file\_ptr);  
Writing acharacter into a file  
putc() is used to write a character into a file   
puts(character-var,file-ptr);  
ClOSING A FILE  
fclose() function close a file.  
fclose(file-ptr);  
fcloseall () is used to close all the opened file at a time  
File Operation  
The following file operation carried out the file  
(1)creation of a new file  
(3)writing a file  
(4)closing a file

Before performing any type of operation we must have to open the file.c, language   
communicate with file using A new type called file pointer.  
Operation with fopen()  
File pointer=fopen(“FILE NAME”,”mode of open”);  
If fopen() unable to open a file then it will return NULL to the file-pointer.  
  
Lecture Note: 30  
Reading and writing a characters from/to a file  
fgetc() is used for reading a character from the file  
Syntax:   
character variable= fgetc(file pointer);  
fputc() is used to writing a character to a file  
Syntax:  
fputc(character,file\_pointer);  
/\*Program to copy a file to another\*/  
#include<stdio.h>  
void main()  
{  
FILE \*fs,\*fd;  
char ch;  
If(fs=fopen(“scr.txt”,”r”)==0)  
{  
printf(“sorry....The source file cannot be opened”);  
return;  
}  
If(fd=fopen(“dest.txt”,”w”)==0)  
{  
printf(“Sorry.....The destination file cannot be opened”);  
fclose(fs);  
return;  
}  
while(ch=fgets(fs)!=EOF)  
fputc(ch,fd);  
fcloseall();  
}  
Reading and writing a string from/to a file  
getw() is used for reading a string from the file  
Syntax:  
gets(file pointer);  
putw() is used to writing a character to a file  
Syntax:   
fputs(integer,file\_pointer);  
#include<stdio.h>  
#include<stdlib.h>  
void main()  
{  
FILE \*fp;  
int word;  
/\*place the word in a file\*/  
fp=fopen(“dgt.txt”,”wb”);  
If(fp==NULL)  
{  
printf(“Error opening file”);  
exit(1);  
}  
word=94;  
putw(word,fp);  
If(ferror(fp))

printf(“Error writing to file\n”);  
else  
printf(“Successful write\n”);  
fclose(fp);  
/\*reopen the file\*/  
fp=fopen(“dgt.txt”,”rb”);  
If(fp==NULL)  
{  
printf(“Error opening file”);  
exit(1);  
}  
/\*extract the word\*/  
word=getw(fp);  
If(ferror(fp))  
printf(“Error reading file\n”);  
else  
printf(“Successful read:word=%d\n”,word);  
/\*clean up\*/  
fclose(fp);  
}

Reading and writing a string from/to a file  
fgets() is used for reading a string from the file  
Syntax:   
fgets(string, length, file pointer);  
fputs() is used to writing a character to a file  
Syntax:   
fputs(string,file\_pointer);  
#include<string.h>  
#include<stdio.h>  
void main(void)  
{  
FILE\*stream;  
char string[]=”This is a test”;  
char msg[20];  
/\*open a file for update\*/  
stream=fopen(“DUMMY.FIL”,”w+”);  
/\*write a string into the file\*/  
fwrite(string,strlen(string),1,stream);  
/\*seek to the start of the file\*/  
fseek(stream,0,SEEK\_SET);

/\*read a string from the file\*/  
fgets(msg,strlen(string)+1,stream);  
/\*display the string\*/  
printf(“%s”,msg);  
fclose(stream);  
}